



# **Intel® Integrated Performance Primitives for Intel® Architecture**

---

*Reference Manual*

## **Volume 2: Image and Video Processing**

Document Number: A70805-017US

World Wide Web: <http://developer.intel.com>

Version	Version Information	Date
-1001	Documents Intel® Integrated Performance Primitives (Intel® IPP) release 1.0 beta.	07/2000
-1002	Documents Intel IPP 1.0 beta 2 . Alpha composition, color twist, gamma correction, and FFT/ DFT/DCT functions have been added.	09/2000
-1003	Documents Intel IPP 1.0 final release. Includes new functionality: wavelet transforms, computer vision functions, and extended geometric transforms. New data initialization, arithmetic, and color conversion functions have also been added.	02/2001
-1101	Documents Intel IPP release 1.1 beta. JPEG codec functions have been added.	04/2001
-2001	Documents Intel IPP release 2.0 beta. Video processing functions for H.263+ and MPEG-4 decoders and wavelet transform functions for JPEG codec have been added.	08/2001
-2002	Documents Intel IPP release 2.0 gold. Library common functions have been added. New flavors of arithmetic, conversion, filtering, statistics, and JPEG codec functions are included.	11/2001
-3001	Documents Intel IPP 3.0 pre-beta. Function flavors to support compatibility layer with domain library have been added. JPEG200 codec functions have been extended. Video decoding functions have been added.	04/2002
-3002	Documents Intel IPP 3.0 beta release.	06/2002
-3003	Documents Intel IPP 3.0 beta update. Video encoding functions were added.	09/2002
-3004	Documents Intel IPP 3.0. The set of MPEG-4 video processing functions was extended.	11/2002
-4001	Documents Intel IPP 4.0 beta. New functions for cross-architecture development added.	05/2003
-4002	Documents Intel IPP 4.0 release.	10/2003
-013	Documents Intel IPP 4.1 beta release.	04/2004
-014	Documents Intel IPP 4.1 release. Added new color conversion functions and flavors for computer vision functions. Updated descriptions of H.264 video coding functions.	07/2004
-015	Documents Intel IPP 5.0 beta release. Added new image support and arithmetic functions, as well as morphological, filtering, and video coding functions. Considerably extended the set of color conversion, computer vision, and image transform functions.	04/2005
-016	Documents Intel IPP 5.0 release. Added new data exchange, image color conversion, filtering, image statistics, image geometric transform, and computer vision functions and function flavors.	08/2005

Version	Version Information	Date
-017	Documents Intel IPP 5.1 release. Added advanced morphology, deconvolution, new computer vision, image arithmetic and geometric transform, and video coding functions, new FFT and DFT function flavors.	03/2006

The information in this manual is subject to change without notice and Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document. This document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. The information in this document is provided in connection with Intel products and should not be construed as a commitment by Intel Corporation.

EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The software described in this document may contain software defects which may cause the product to deviate from published specifications. Current characterized software defects are available on request.

MPEG-1, MPEG-2, MPEG-4, H.261, H.263, H.264, MP3, DV, MJPEG, AC3, and AAC are international standards promoted by ISO, IEC, ITU, ETSI and other organizations. Implementations of these standards, or the standard enabled platforms may require licenses from various entities, including Intel Corporation.

Intel, the Intel logo, Intel SpeedStep, Intel NetBurst, Intel NetStructure, MMX, Intel386, Intel486, Celeron, Intel Centrino, Intel Xeon, Intel XScale, Itanium, Pentium, Pentium II Xeon, Pentium III Xeon, Pentium M, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

\* Other names and brands may be claimed as the property of others.

Copyright © 2000-2006, Intel Corporation.

# Contents

---

## Chapter 1 Overview

About This Software .....	1-1
Hardware and Software Requirements .....	1-2
Platforms Supported.....	1-2
Cross Architecture Alignment .....	1-2
Cross Architecture Overview .....	1-2
API Changes in Version 5.0.....	1-3
Technical Support.....	1-4
Intel® IPP Code Samples .....	1-4
About This Manual .....	1-5
Manual Organization .....	1-5
Function Descriptions .....	1-6
Audience for This Manual .....	1-7
Online Version.....	1-7
Related Publications .....	1-7
Notational Conventions .....	1-7
Font Conventions .....	1-7
Naming Conventions .....	1-7

## Chapter 2 Intel® Integrated Performance Primitives Concepts

Basic Features .....	2-1
Function Naming .....	2-2
Data-Domain .....	2-2
Name .....	2-3



	Data Types .....	2-3
	Descriptor .....	2-5
	Arguments .....	2-6
	Function Prototypes in Intel IPP .....	2-6
	Integer Result Scaling .....	2-8
	Error Reporting .....	2-8
	Structures and Enumerators .....	2-13
	Image Data Types and Ranges .....	2-18
	Major Operation Models .....	2-20
	Neighborhood operations .....	2-20
	Regions of Interest in Intel IPP .....	2-20
	Tiled Image Processing .....	2-24
<b>Chapter 3</b>	<b>Support Functions</b>	
	Version Information Function .....	3-2
	GetLibVersion .....	3-2
	Status Information Function .....	3-4
	ippGetStatusString .....	3-4
	Memory Allocation Functions .....	3-5
	Malloc .....	3-5
	Free .....	3-7
<b>Chapter 4</b>	<b>Image Data Exchange and Initialization Functions</b>	
	Convert .....	4-2
	Scale .....	4-6
	Set .....	4-9
	Copy .....	4-11
	CopyConstBorder .....	4-15
	CopyReplicateBorder .....	4-18
	CopyWrapBorder .....	4-20
	CopySubpix .....	4-23
	CopySubpixIntersect .....	4-24
	Dup .....	4-27
	Transpose .....	4-28

SwapChannels .....	4-30
AddRandUniform_Direct .....	4-31
AddRandGauss_Direct .....	4-34
ImageJaehne .....	4-35
ImageRamp .....	4-37
SampleLine .....	4-39
ZigzagFwd8x8 .....	4-40
ZigzagInv8x8 .....	4-41

## Chapter 5    **Image Arithmetic and Logical Operations**

Arithmetic Operations .....	5-4
Add .....	5-4
AddC .....	5-7
AddSquare .....	5-11
AddProduct .....	5-13
AddWeighted .....	5-14
Mul .....	5-16
MulC .....	5-19
MulScale .....	5-23
MulCScale .....	5-25
Sub .....	5-27
SubC .....	5-30
Div .....	5-34
DivC .....	5-37
Abs .....	5-40
AbsDiff .....	5-42
AbsDiffC .....	5-43
Sqr .....	5-44
Sqrt .....	5-46
Ln .....	5-49
Exp .....	5-52
Complement .....	5-54
Logical Operations .....	5-55
And .....	5-55

AndC .....	5-57
Not .....	5-59
Or .....	5-61
OrC .....	5-62
Xor .....	5-65
XorC .....	5-67
LShiftC .....	5-69
RShiftC .....	5-72
Alpha Composition .....	5-75
AlphaComp .....	5-76
AlphaCompC .....	5-78
AlphaPremul .....	5-81
AlphaPremulC .....	5-83

## Chapter 6 Image Color Conversion

Gamma Correction .....	6-7
CIE Chromaticity Diagram and Color Gamut .....	6-7
Color Models .....	6-8
RGB Color Model .....	6-9
CMYK Color Model .....	6-11
YUV Color Model .....	6-11
YCbCr and YCCK Color Models .....	6-13
PhotoYCC Color Model .....	6-14
YCoCg Color Models .....	6-16
HSV, and HLS Color Models .....	6-17
CIE XYZ Color Model .....	6-20
CIE LUV and CIE Lab Color Models .....	6-21
Image Downsampling .....	6-24
RGB Image Formats .....	6-26
Pixel and Planar Image Formats .....	6-27
Color Model Conversion .....	6-31
RGBToYUV .....	6-31
YUVToRGB .....	6-33
RGBToYUV422 .....	6-34

YUV422ToRGB .....	6-36
RGBToYUV420 .....	6-38
YUV420ToRGB .....	6-39
YUV420ToBGR.....	6-41
YUV420ToRGB565	
YUV420ToRGB555	
YUV420ToRGB444 .....	6-42
YUV420ToRGB565Dither	
YUV420ToRGB555Dither	
YUV420ToRGB444Dither .....	6-43
BGR565ToYUV420	
BGR555ToYUV420 .....	6-44
YUV420ToBGR565	
YUV420ToBGR555	
YUV420ToBGR444 .....	6-46
YUV420ToBGR565Dither	
YUV420ToBGR555Dither	
YUV420ToBGR444Dither .....	6-47
RGBToYCbCr .....	6-48
YCbCrToRGB .....	6-50
YCbCrToBGR .....	6-51
YCbCrToRGB565	
YCbCrToRGB555	
YCbCrToRGB444 .....	6-52
YCbCrToRGB565Dither	
YCbCrToRGB555Dither	
YCbCrToRGB444Dither .....	6-54
YCbCrToBGR565	
YCbCrToBGR555	
YCbCrToBGR444 .....	6-55
YCbCrToBGR565Dither	
YCbCrToBGR555Dither	
YCbCrToBGR444Dither .....	6-57
RGBToYCbCr422 .....	6-58
YCbCr422ToRGB .....	6-59
BGRToYCbCr422 .....	6-61

YCbCr422ToBGR .....	6-62
BGR565ToYCbCr422	
BGR555ToYCbCr422 .....	6-63
RGBToCbYCr422	
RGBToCbYCr422Gamma .....	6-65
CbYCr422ToRGB .....	6-66
BGRToCbYCr422 .....	6-67
CbYCr422ToBGR .....	6-68
YCbCr422ToRGB565	
YCbCr422ToRGB555	
YCbCr422ToRGB444 .....	6-69
YCbCr422ToRGB565Dither	
YCbCr422ToRGB555Dither	
YCbCr422ToRGB444Dither .....	6-71
YCbCr422ToBGR565	
YCbCr422ToBGR555	
YCbCr422ToBGR444 .....	6-72
YCbCr422ToBGR565Dither	
YCbCr422ToBGR555Dither	
YCbCr422ToBGR444Dither .....	6-74
RGBToYCbCr420 .....	6-75
YCbCr420ToRGB .....	6-76
YCbCr420ToRGB565	
YCbCr420ToRGB555	
YCbCr420ToRGB444 .....	6-77
YCbCr420ToRGB565Dither	
YCbCr420ToRGB555Dither	
YCbCr420ToRGB444Dither .....	6-79
BGRToYCbCr420 .....	6-80
YCbCr420ToBGR .....	6-81
BGR565ToYCbCr420,	
BGR555ToYCbCr420 .....	6-82
YCbCr420ToBGR565	
YCbCr420ToBGR555	
YCbCr420ToBGR444 .....	6-83
YCbCr420ToBGR565Dither	

YCbCr420ToBGR555Dither	
YCbCr420ToBGR444Dither .....	6-85
BGRToYCrCb420 .....	6-86
BGR565ToYCrCb420	
BGR555ToYCrCb420 .....	6-87
BGRToYCbCr411 .....	6-88
YCbCr411ToBGR .....	6-89
BGR565ToYCbCr411,	
BGR555ToYCbCr411 .....	6-90
YCbCr411ToBGR565,	
YCbCr411ToBGR555 .....	6-92
RGBToXYZ .....	6-93
XYZToRGB .....	6-94
RGBToLUV .....	6-95
LUVToRGB .....	6-97
BGRToLab .....	6-99
LabToBGR .....	6-101
RGBToYCC .....	6-102
YCCToRGB .....	6-104
RGBToHLS .....	6-105
HLSToRGB .....	6-107
BGRToHLS .....	6-109
HLSToBGR .....	6-110
RGBToHSV .....	6-111
HSVToRGB .....	6-113
BGRToYCoCg .....	6-114
SBGRToYCoCg .....	6-115
YCoCgToBGR .....	6-116
YCoCgToSBGR .....	6-117
BGRToYCoCg_Rev .....	6-118
SBGRToYCoCg_Rev .....	6-119
YCoCgToBGR_Rev .....	6-120
YCoCgToSBGR_Rev .....	6-121
Color to Gray Scale Conversion.....	6-123

RGBToGray .....	6-123
ColorToGray .....	6-124
Lookup Table Conversion .....	6-125
LUT .....	6-125
LUT_Linear .....	6-129
LUT_Cubic .....	6-132
LUTPalette .....	6-137
Reducing Bit Resolution .....	6-138
ReduceBits .....	6-138
Format Conversion .....	6-140
YCbCr422 .....	6-141
YCbCr422ToYCrCb422 .....	6-142
YCbCr422ToCbYCr422 .....	6-143
YCbCr422ToYCbCr420 .....	6-144
YCbCr422ToYCrCb420 .....	6-146
YCbCr422ToYCbCr411 .....	6-147
YCrCb422ToYCbCr422 .....	6-149
YCrCb422ToYCbCr420 .....	6-150
YCrCb422ToYCbCr411 .....	6-151
CbYCr422ToYCbCr422 .....	6-152
CbYCr422ToYCbCr420 .....	6-153
CbYCr422ToYCrCb420 .....	6-154
CbYCr422ToYCbCr411 .....	6-155
YCbCr420 .....	6-156
YCbCr420ToYCbCr422 .....	6-157
YCbCr420ToYCbCr422_Filter .....	6-159
YCbCr420ToCbYCr422 .....	6-161
YCbCr420ToYCrCb420 .....	6-162
YCbCr420ToYCrCb420_Filter .....	6-163
YCbCr420ToYCbCr411 .....	6-165
YCrCb420ToYCbCr422 .....	6-166
YCrCb420ToYCbCr422_Filter .....	6-167
YCrCb420ToCbYCr422 .....	6-168
YCrCb420ToYCbCr420 .....	6-169

YCrCb420ToYCbCr411 .....	6-170
YCbCr411 .....	6-171
YCbCr411ToYCbCr422 .....	6-173
YCbCr411ToYCrCb422 .....	6-174
YCbCr411ToYCbCr420 .....	6-175
YCbCr411ToYCrCb420 .....	6-177
Color Twist .....	6-178
ColorTwist .....	6-179
ColorTwist32f .....	6-181
Gamma Correction .....	6-183
GammaFwd .....	6-183
GammaInv .....	6-186

## Chapter 7    **Threshold and Compare Operations**

Thresholding .....	7-2
Threshold .....	7-2
Threshold_GT .....	7-5
Threshold_LT .....	7-7
Threshold_Val .....	7-9
Threshold_GTVal .....	7-11
Threshold_LTVal .....	7-13
Threshold_LTValGTVal .....	7-16
Compare Operations .....	7-19
Compare .....	7-19
CompareC .....	7-20
CompareEqualEps .....	7-23
CompareEqualEpsC .....	7-24

## Chapter 8    **Morphological Operations**

Flat Structuring Elements for Grayscale Image .....	8-6
Dilate3x3 .....	8-7
Erode3x3 .....	8-9
Dilate .....	8-11
Erode .....	8-13



MorphologyInitAlloc .....	8-15
MorphologyFree .....	8-16
MorphologyInit .....	8-17
MorphologyGetSize .....	8-18
DilateBorderReplicate .....	8-19
ErodeBorderReplicate .....	8-21
MorphAdvInitAlloc .....	8-22
MorphAdvFree .....	8-24
MorphAdvInit .....	8-24
MorphAdvGetSize .....	8-26
MorphOpenBorder .....	8-27
MorphCloseBorder .....	8-28
MorphTophatBorder .....	8-30
MorphBlackhatBorder .....	8-31
MorphGradientBorder .....	8-33
MorphGrayInitAlloc .....	8-35
MorphGrayFree .....	8-36
MorphGrayInit .....	8-37
MorphGrayGetSize .....	8-38
GrayDilateBorder .....	8-39
GrayErodeBorder .....	8-40
MorphReconstructGetBufferSize .....	8-42
MorphReconstructDilate .....	8-43
MorphReconstructErode .....	8-46

## **Chapter 9    Filtering Functions**

Borders .....	9-5
FilterBox .....	9-8
SumWindowRow .....	9-10
SumWindowColumn .....	9-11
FilterMin .....	9-12
FilterMax .....	9-15
FilterMinGetBufferSize .....	9-16
FilterMaxGetBufferSize .....	9-17

FilterMinBorderReplicate .....	9-18
FilterMaxBorderReplicate .....	9-20
Median Filters .....	9-22
FilterMedian .....	9-23
FilterMedianHoriz .....	9-25
FilterMedianVert .....	9-26
FilterMedianCross .....	9-28
FilterMedianColor .....	9-29
General Linear Filters .....	9-31
Filter .....	9-31
Filter32f .....	9-34
Separable Filters.....	9-36
FilterColumn .....	9-36
FilterColumn32f .....	9-38
FilterRow .....	9-39
FilterRow32f .....	9-42
FilterRowBorderPipelineGetBufferSize .....	9-43
FilterRowBorderPipelineGetBufferSize_Low .....	9-44
FilterColumnPipelineGetBufferSize .....	9-45
FilterColumnPipelineGetBufferSize_Low .....	9-46
FilterRowBorderPipeline .....	9-47
FilterRowBorderPipeline_Low .....	9-50
FilterColumnPipeline .....	9-52
FilterColumnPipeline_Low .....	9-54
Wiener Filters.....	9-57
FilterWienerGetBufferSize .....	9-57
FilterWiener .....	9-58
Convolution .....	9-61
ConvFull .....	9-61
ConvValid .....	9-64
Deconvolution .....	9-67
DeconvFFTInitAlloc .....	9-67
DeconvFFTFree .....	9-68
DeconvFFT .....	9-69

DeconvLRInitAlloc .....	9-70
DeconvLRFree .....	9-71
DeconvLR .....	9-71
Fixed Filters .....	9-73
FilterPrewittHoriz .....	9-74
FilterPrewittVert .....	9-75
FilterScharrHoriz .....	9-78
FilterScharrVert .....	9-79
FilterSobelHoriz, FilterSobelHorizMask .....	9-80
FilterSobelVert, FilterSobelVertMask .....	9-82
FilterSobelHorizSecond .....	9-84
FilterSobelVertSecond .....	9-85
FilterSobelCross .....	9-87
FilterRobertsDown .....	9-88
FilterRobertsUp .....	9-89
FilterLaplace .....	9-91
FilterGauss .....	9-92
FilterHipass .....	9-94
FilterLowpass .....	9-95
FilterSharpen .....	9-97
Fixed Filters with Border .....	9-99
FilterScharrHorizGetBufferSize .....	9-99
FilterScharrVertGetBufferSize .....	9-100
FilterSobelHorizGetBufferSize .....	9-101
FilterSobelVertGetBufferSize, FilterSobelNegVertGetBufferSize .....	9-102
FilterSobelHorizSecondGetBufferSize .....	9-103
FilterSobelVertSecondGetBufferSize .....	9-104
FilterSobelCrossGetBufferSize .....	9-105
FilterLaplacianGetBufferSize .....	9-106
FilterLowpassGetBufferSize .....	9-107
GenSobelKernel .....	9-108
FilterScharrHorizBorder .....	9-109
FilterScharrVertBorder .....	9-110

FilterSobelHorizBorder .....	9-112
FilterSobelVertBorder, FilterSobelNegVertBorder, .....	9-114
FilterSobelHorizSecondBorder .....	9-116
FilterSobelVertSecondBorder .....	9-118
FilterSobelCrossBorder .....	9-120
FilterLaplacianBorder .....	9-122
FilterLowpassBorder .....	9-124

## Chapter 10 Image Linear Transforms

Fourier Transforms .....	10-4
Real - Complex Packed (RCPack2D) Format .....	10-5
FFTInitAlloc .....	10-6
FFTFree .....	10-7
FFTGetBufSize .....	10-8
FFTFwd .....	10-9
FFTInv .....	10-14
DFTInitAlloc .....	10-16
DFTFree .....	10-17
DFTGetBufSize .....	10-18
DFTFwd .....	10-19
DFTInv .....	10-22
MulPack .....	10-25
MulPackConj .....	10-28
Magnitude .....	10-30
MagnitudePack .....	10-31
Phase .....	10-32
PhasePack .....	10-34
PolarToCart.....	10-35
PackToCplxExtend .....	10-36
Windowing Functions.....	10-37
WinBartlett, WinBartlettSep .....	10-38
WinHamming,	

WinHammingSep .....	10-40
Discrete Cosine Transforms .....	10-42
DCTFwdInitAlloc .....	10-42
DCTInvInitAlloc .....	10-43
DCTFwdFree .....	10-44
DCTInvFree .....	10-45
DCTFwdGetBufSize .....	10-45
DCTInvGetBufSize .....	10-46
DCTFwd .....	10-47
DCTInv .....	10-49
DCT8x8Fwd .....	10-50
DCT8x8Inv .....	10-52
DCT8x8FwdLS .....	10-54
DCT8x8InvLSClip .....	10-55
DCT8x8Inv_2x2, DCT8x8Inv_4x4 .....	10-56

## Chapter 11 Image Statistics Functions

Sum .....	11-5
Integral .....	11-7
SqrIntegral .....	11-9
TiltedIntegral .....	11-11
TiltedSqrIntegral .....	11-13
Mean .....	11-15
Mean_StdDev .....	11-18
RectStdDev .....	11-20
TiltedRectStdDev .....	11-22
HistogramRange .....	11-24
HistogramEven .....	11-27
CountInRange .....	11-30
Min .....	11-31
MinIndx .....	11-33
Max .....	11-34
MaxIndx .....	11-36

MinMax .....	11-37
MinMaxIndx .....	11-39
Image Moments .....	11-41
MomentInitAlloc .....	11-42
MomentFree .....	11-43
MomentGetStateSize .....	11-44
MomentInit .....	11-45
Moments .....	11-45
GetSpatialMoment .....	11-47
GetNormalizedSpatialMoment .....	11-48
GetCentralMoment .....	11-50
GetNormalizedCentralMoment .....	11-51
GetHuMoments .....	11-52
Image Norms .....	11-54
Norm_Inf .....	11-54
Norm_L1 .....	11-56
Norm_L2 .....	11-60
NormDiff_Inf .....	11-62
NormDiff_L1 .....	11-65
NormDiff_L2 .....	11-68
NormRel_Inf .....	11-71
NormRel_L1 .....	11-73
NormRel_L2 .....	11-76
Image Quality Index .....	11-79
QualityIndex .....	11-80
Image Proximity Measures .....	11-82
SqrDistanceFull_Norm .....	11-84
SqrDistanceSame_Norm .....	11-86
SqrDistanceValid_Norm .....	11-88
CrossCorrFull_Norm .....	11-90
CrossCorrSame_Norm .....	11-92
CrossCorrValid_Norm .....	11-94
CrossCorrFull_NormLevel .....	11-96
CrossCorrSame_NormLevel .....	11-98

CrossCorrValid_NormLevel .....	11-100
--------------------------------	--------

## **Chapter 12 Image Geometric Transforms**

ROI Processing in Geometric Transforms.....	12-4
Resize .....	12-5
ResizeCenter .....	12-8
ResizeSqrPixel .....	12-12
ResizeSqrPixelGetBufSize .....	12-15
GetResizeFract .....	12-16
ResizeShift .....	12-17
ResizeYUV422 .....	12-20
Mirror .....	12-21
Remap .....	12-23
Rotate .....	12-26
GetRotateShift .....	12-30
AddRotateShift .....	12-31
GetRotateQuad .....	12-32
GetRotateBound .....	12-33
RotateCenter .....	12-34
Shear .....	12-36
GetShearQuad .....	12-39
GetShearBound .....	12-40
WarpAffine .....	12-41
WarpAffineBack .....	12-45
WarpAffineQuad .....	12-48
GetAffineQuad .....	12-51
GetAffineBound .....	12-52
GetAffineTransform .....	12-53
WarpPerspective .....	12-54
WarpPerspectiveBack .....	12-57
WarpPerspectiveQuad .....	12-60
GetPerspectiveQuad .....	12-62
GetPerspectiveBound .....	12-63
GetPerspectiveTransform .....	12-64

WarpBilinear .....	12-66
WarpBilinearBack .....	12-69
WarpBilinearQuad .....	12-72
GetBilinearQuad .....	12-75
GetBilinearBound .....	12-76
GetBilinearTransform .....	12-77

## Chapter 13 Wavelet Transforms

WTFwdInitAlloc .....	13-5
WTFwdFree .....	13-7
WTFwdGetBufSize .....	13-7
WTFwd .....	13-8
WTInvInitAlloc .....	13-13
WTInvFree .....	13-15
WTInvGetBufSize .....	13-16
WTInv .....	13-17

## Chapter 14 Computer Vision

Using ippiAdd for Background Differencing .....	14-3
Feature Detection Functions .....	14-4
Corner Detection .....	14-5
Canny Edge Detector .....	14-5
Stage 1: Differentiation .....	14-5
Stage 2: Non-Maximum Suppression .....	14-6
Stage 3: Edge Thresholding .....	14-6
CannyGetSize .....	14-7
Canny .....	14-8
EigenValsVecsGetBufferSize .....	14-9
EigenValsVecs .....	14-10
MinEigenValGetBufferSize .....	14-13
MinEigenVal .....	14-14
Distance Transform Functions .....	14-16
DistanceTransform .....	14-16
GetDistanceTransformMask .....	14-19



Image Gradients .....	14-21
GradientColorToGray .....	14-21
Flood Fill Functions .....	14-22
FloodFillGetSize .....	14-23
FloodFillGetSize_Grad .....	14-24
FloodFill .....	14-25
FloodFill_Grad .....	14-27
FloodFill_Range .....	14-29
Motion Analysis and Object Tracking.....	14-31
Motion Template Functions.....	14-31
Motion Representation .....	14-31
Updating MHI Images .....	14-32
UpdateMotionHistory .....	14-33
Optical Flow .....	14-34
OpticalFlowPyrLKInitAlloc .....	14-35
OpticalFlowPyrLKFree .....	14-36
OpticalFlowPyrLK .....	14-36
Pyramids Functions .....	14-39
PyrDownGetBufSize .....	14-40
PyrUpGetBufSize .....	14-41
PyrDown .....	14-42
PyrUp .....	14-43
Universal Pyramids.....	14-45
PyramidInitAlloc .....	14-45
PyramidFree .....	14-46
PyramidLayerDownInitAlloc .....	14-47
PyramidLayerDownFree .....	14-49
PyramidLayerUpInitAlloc .....	14-49
PyramidLayerUpFree .....	14-51
GetPyramidDownROI .....	14-52
GetPyramidUpROI .....	14-53
PyramidLayerDown .....	14-54
PyramidLayerUp .....	14-56
Example of Using General Pyramid Functions .....	14-58

Pattern Recognition .....	14-60
Object Detection Using Haar-like Features .....	14-60
HaarClassifierInitAlloc .....	14-62
TiltedHaarClassifierInitAlloc .....	14-63
HaarClassifierFree .....	14-65
GetHaarClassifierSize .....	14-65
TiltHaarFeatures .....	14-66
ApplyHaarClassifier .....	14-67
ApplyMixedHaarClassifier .....	14-69
Camera Calibration and 3D Reconstruction .....	14-72
Correction of Camera Lens Distortion.....	14-72
UndistortGetSize .....	14-72
UndistortRadial .....	14-73
CreateMapCameraUndistort .....	14-75

## Chapter 15 Image Compression Functions

Support Functions .....	15-2
ippjGetLibVersion .....	15-2
Color Conversion Functions .....	15-3
RGBToY_JPEG .....	15-4
BGRTToY_JPEG .....	15-5
RGBToYCbCr_JPEG .....	15-6
YCbCrToRGB_JPEG .....	15-7
RGB565ToYCbCr_JPEG, RGB555ToYCbCr_JPEG .....	15-8
YCbCrToRGB565_JPEG, YCbCrToRGB555_JPEG .....	15-9
BGRTToYCbCr_JPEG .....	15-10
YCbCrToBGR_JPEG .....	15-11
BGR565ToYCbCr_JPEG, BGR555ToYCbCr_JPEG .....	15-12
YCbCrToBGR565_JPEG, YCbCrToBGR555_JPEG .....	15-13
CMYKToYCCK_JPEG .....	15-15
YCCKToCMYK_JPEG .....	15-16

Combined Color Conversion Functions .....	15-18
RGBToYCbCr444LS_MCU .....	15-19
RGBToYCbCr422LS_MCU .....	15-20
RGBToYCbCr411LS_MCU .....	15-21
BGRToYCbCr444LS_MCU .....	15-22
BGR565ToYCbCr444LS_MCU, BGR555ToYCbCr444LS_MCU .....	15-23
BGRToYCbCr422LS_MCU .....	15-24
BGR565ToYCbCr422LS_MCU, BGR555ToYCbCr422LS_MCU .....	15-25
BGRToYCbCr411LS_MCU .....	15-26
BGR565ToYCbCr411LS_MCU, BGR555ToYCbCr411LS_MCU .....	15-27
CMYKToYCK444LS_MCU .....	15-28
CMYKToYCK422LS_MCU .....	15-29
CMYKToYCK411LS_MCU .....	15-30
YCbCr444ToRGBLS_MCU .....	15-31
YCbCr422ToRGBLS_MCU .....	15-32
YCbCr411ToRGBLS_MCU .....	15-33
YCbCr444ToBGRLS_MCU .....	15-34
YCbCr444ToBGR565LS_MCU, YCbCr444ToBGR555LS_MCU .....	15-35
YCbCr422ToBGRLS_MCU .....	15-36
YCbCr422ToBGR565LS_MCU, YCbCr422ToBGR555LS_MCU .....	15-37
YCbCr411ToBGRLS_MCU .....	15-38
YCbCr411ToBGR565LS_MCU, YCbCr411ToBGR555LS_MCU .....	15-39
YCK444ToCMYKLS_MCU .....	15-40
YCK422ToCMYKLS_MCU .....	15-41
YCK411ToCMYKLS_MCU .....	15-42
Quantization Functions .....	15-43
QuantFwdRawTableInit_JPEG .....	15-43
QuantFwdTableInit_JPEG .....	15-44
QuantFwd8x8_JPEG .....	15-45

---

QuantInvTableInit_JPEG .....	15-46
QuantInv8x8_JPEG .....	15-46
Combined DCT Functions .....	15-48
DCTQuantFwd8x8_JPEG .....	15-48
DCTQuantFwd8x8LS_JPEG .....	15-49
DCTQuantInv8x8_JPEG .....	15-50
DCTQuantInv8x8LS_JPEG .....	15-51
Level Shift Functions .....	15-52
Sub128_JPEG .....	15-52
Add128_JPEG .....	15-53
Sampling Functions .....	15-54
SampleDownH2V1_JPEG .....	15-55
SampleDownH2V2_JPEG .....	15-56
SampleDownRowH2V1_Box_JPEG .....	15-57
SampleDownRowH2V2_Box_JPEG .....	15-58
SampleUpH2V1_JPEG .....	15-59
SampleUpH2V2_JPEG .....	15-61
SampleUpRowH2V1_Triangle_JPEG .....	15-62
SampleUpRowH2V2_Triangle_JPEG .....	15-63
SampleDown444LS_MCU .....	15-64
SampleDown422LS_MCU .....	15-65
SampleDown411LS_MCU .....	15-66
SampleUp444LS_MCU .....	15-67
SampleUp422LS_MCU .....	15-68
SampleUp411LS_MCU .....	15-69
Planar-to-Pixel and Pixel-to-Planar Conversion Functions .....	15-70
Split422LS_MCU .....	15-70
Join422LS_MCU .....	15-71
Huffman Codec Functions .....	15-72
EncodeHuffmanRawTableInit_JPEG .....	15-74
EncodeHuffmanSpecGetBufSize_JPEG .....	15-75
EncodeHuffmanSpecInit_JPEG .....	15-76
EncodeHuffmanSpecInitAlloc_JPEG .....	15-77
EncodeHuffmanSpecFree_JPEG .....	15-78

EncodeHuffmanStateGetBufSize_JPEG .....	15-78
EncodeHuffmanStateInit_JPEG .....	15-79
EncodeHuffmanStateInitAlloc_JPEG .....	15-80
EncodeHuffmanStateFree_JPEG .....	15-80
EncodeHuffman8x8_JPEG .....	15-81
EncodeHuffman8x8_Direct_JPEG .....	15-82
GetHuffmanStatistics8x8_JPEG .....	15-83
GetHuffmanStatistics8x8_DCFirst_JPEG .....	15-84
GetHuffmanStatistics8x8_ACFirst_JPEG .....	15-85
GetHuffmanStatistics8x8_ACRefine_JPEG .....	15-86
EncodeHuffman8x8_DCFirst_JPEG .....	15-87
EncodeHuffman8x8_DCRefine_JPEG .....	15-88
EncodeHuffman8x8_ACFirst_JPEG .....	15-90
EncodeHuffman8x8_ACRefine_JPEG .....	15-91
DecodeHuffmanSpecGetBufSize_JPEG .....	15-92
DecodeHuffmanSpecInit_JPEG .....	15-93
DecodeHuffmanSpecInitAlloc_JPEG .....	15-94
DecodeHuffmanSpecFree_JPEG .....	15-95
DecodeHuffmanStateGetBufSize_JPEG .....	15-96
DecodeHuffmanStateInit_JPEG .....	15-96
DecodeHuffmanStateInitAlloc_JPEG .....	15-97
DecodeHuffmanStateFree_JPEG .....	15-98
DecodeHuffman8x8_JPEG .....	15-98
DecodeHuffman8x8_Direct_JPEG .....	15-100
DecodeHuffman8x8_DCFirst_JPEG .....	15-101
DecodeHuffman8x8_DCRefine_JPEG .....	15-103
DecodeHuffman8x8_ACFirst_JPEG .....	15-104
DecodeHuffman8x8_ACRefine_JPEG .....	15-105
Functions for Lossless JPEG Coding .....	15-107
DiffPredFirstRow_JPEG .....	15-107
DiffPredRow_JPEG .....	15-108
ReconstructPredFirstRow_JPEG .....	15-109
ReconstructPredRow_JPEG .....	15-110
GetHuffmanStatisticsOne_JPEG .....	15-111

---

EncodeHuffmanOne_JPEG .....	15-112
DecodeHuffmanOne_JPEG .....	15-113
Wavelet Transform Functions.....	15-114
Low-Level Operations.....	15-115
WTFwdRow_B53_JPEG2K .....	15-117
WTInvRow_B53_JPEG2K .....	15-118
WTFwdCol_B53_JPEG2K .....	15-120
WTFwdColLift_B53_JPEG2K .....	15-121
WTInvCol_B53_JPEG2K .....	15-123
WTInvColLift_B53_JPEG2K .....	15-124
WTFwdRow_D97_JPEG2K .....	15-126
WTInvRow_D97_JPEG2K .....	15-127
WTFwdCol_D97_JPEG2K .....	15-129
WTFwdColLift_D97_JPEG2K .....	15-130
WTInvCol_D97_JPEG2K .....	15-132
WTInvColLift_D97_JPEG2K .....	15-133
Tile-Oriented Transforms .....	15-135
WTGetBufSize_B53_JPEG2K .....	15-135
WTFwd_B53_JPEG2K .....	15-136
WTInv_B53_JPEG2K .....	15-138
WTGetBufSize_D97_JPEG2K .....	15-139
WTFwd_D97_JPEG2K .....	15-140
WTInv_D97_JPEG2K .....	15-142
JPEG2000 Entropy Coding and Decoding Functions .....	15-144
EncodeInitAlloc_JPEG2K .....	15-145
EncodeFree_JPEG2K .....	15-146
EncodeLoadCodeBlock_JPEG2K .....	15-146
EncodeStoreBits_JPEG2K .....	15-149
EncodeGetTermPassLen_JPEG2K .....	15-151
EncodeGetRate_JPEG2K .....	15-152
EncodeGetDist_JPEG2K .....	15-153
DecodeGetBufSize_JPEG2K .....	15-154
DecodeCodeBlock_JPEG2K .....	15-154
DecodeCBProgrGetStateSize_JPEG2K .....	15-156

DecodeCBProgrInit_JPEG2K .....	15-157
DecodeCBProgrInitAlloc_JPEG2K .....	15-157
DecodeCBProgrFree_JPEG2K .....	15-158
DecodeCBProgrAttach_JPEG2K .....	15-159
DecodeCBProgrSetPassCounter_JPEG2K .....	15-160
DecodeCBProgrGetPassCounter_JPEG2K .....	15-161
DecodeCBProgrGetCurBitPlane_JPEG2K .....	15-162
DecodeCBProgrStep_JPEG2K .....	15-163
Component Transform Functions .....	15-164
RCTFwd_JPEG2K .....	15-164
RCTInv_JPEG2K .....	15-166
ICTFwd_JPEG2K .....	15-167
ICTInv_JPEG2K .....	15-169

## **Chapter 16 Video Coding**

General Functions .....	16-2
Structures and Enumerators .....	16-3
Variable Length Decoding .....	16-11
HuffmanTableInitAlloc .....	16-18
HuffmanRunLevelTableInitAlloc .....	16-19
DecodeHuffmanOne .....	16-20
DecodeHuffmanPair .....	16-21
HuffmanTableFree .....	16-22
Motion Compensation .....	16-23
MC16x16 .....	16-24
MC16x8 .....	16-25
MC8x16 .....	16-26
MC8x8 .....	16-27
MC8x4 .....	16-28
MC4x8 .....	16-29
MC4x4 .....	16-30
MC2x4 .....	16-31
MC4x2 .....	16-32
MC2x2 .....	16-33

MC16x4 .....	16-34
MC16x8UV .....	16-35
MC16x16B .....	16-36
MC16x8B .....	16-38
MC8x16B .....	16-39
MC8x8B .....	16-40
MC8x4B .....	16-41
MC4x8B .....	16-42
MC4x4B .....	16-44
MC2x4B .....	16-45
MC4x2B .....	16-46
MC2x2B .....	16-47
MC16x4B .....	16-48
MC16x8UVB .....	16-50
Copy8x8, Copy16x16 .....	16-51
Copy8x4HP, Copy8x8HP, Copy16x8HP, Copy16x16HP .....	16-52
InterpolateAverage8x4, InterpolateAverage8x8, InterpolateAverage16x8, InterpolateAverage16x16 .....	16-53
Add8x8 .....	16-54
Add8x8HP .....	16-55
AddC8x8 .....	16-56
Average8x8, Average16x16 .....	16-57
Motion Estimation.....	16-58
Difference Evaluation .....	16-59
GetDiff16x16 .....	16-60
GetDiff16x8 .....	16-61
GetDiff8x8 .....	16-62
GetDiff8x16 .....	16-64



GetDiff8x4 .....	16-65
GetDiff4x4 .....	16-66
GetDiff16x16B .....	16-67
GetDiff16x8B .....	16-69
GetDiff8x8B .....	16-70
GetDiff8x16B .....	16-71
GetDiff8x4B .....	16-72
Sub8x8, Sub16x16 .....	16-73
SubSAD8x8 .....	16-74
Sum of Squares of Differences Evaluation.....	16-75
SqrDiff16x16 .....	16-76
SqrDiff16x16B .....	16-77
SSD8x8 .....	16-78
SSD4x4 .....	16-79
Block Variance and Mean Evaluation.....	16-80
VarMean8x8 .....	16-80
Evaluation of Variances and Means of Blocks of Difference Between Two Blocks .....	16-82
VarMeanDiff16x16 .....	16-82
VarMeanDiff16x8 .....	16-83
Block Variance Evaluation.....	16-85
Variance16x16 .....	16-85
Evaluation of Block Deviation.....	16-86
MeanAbsDev16x16 .....	16-86
Edges Detection.....	16-87
EdgesDetect16x16 .....	16-87
SAD Functions .....	16-88
SAD16x16 .....	16-88
SAD16x8 .....	16-89
SAD8x8 .....	16-90
SAD4x4 .....	16-91
SAD16x16Blocks8x8 .....	16-92
SAD16x16Blocks4x4 .....	16-93

FrameFieldSAD16x16 .....	16-94
Sum of Differences Evaluation .....	16-96
SumsDiff16x16Blocks4x4 .....	16-96
SumsDiff8x8Blocks4x4 .....	16-98
Scanning Functions.....	16-99
ScanInv .....	16-99
ScanFwd .....	16-101
ZigzagInvClassical_Compact,	
ZigzagInvHorizontal_Compact,	
ZigzagInvVertical_Compact .....	16-102
Color Conversion .....	16-104
CbYCr422ToYCbCr420_Rotate .....	16-104
ResizeCCRotate .....	16-106
Video Processing.....	16-109
DeinterlaceFilterTriangle .....	16-109
MPEG-1 and MPEG-2 .....	16-111
Structures and Enumerations.....	16-111
Video Data Decoding .....	16-115
Variable Length Decoding .....	16-118
ReconstructDCTBlock_MPEG1 .....	16-119
ReconstructDCTBlockIntra_MPEG1 .....	16-120
ReconstructDCTBlock_MPEG2 .....	16-122
ReconstructDCTBlockIntra_MPEG2 .....	16-125
Inverse Quantization.....	16-126
QuantInvIntra_MPEG2 .....	16-127
QuantInv_MPEG2 .....	16-128
Inverse Discrete Cosine Transformation.....	16-129
DCT8x8Inv_AANTransposed_16s_C1R .....	16-129
DCT8x8Inv_AANTransposed_16s8u_C1R.....	16-130
DCT8x8Inv_AANTransposed_16s_P2C2R .....	16-131
DCT8x8Inv_AANTransposed_16s8u_P2C2R .....	16-132
Motion Compensation .....	16-133
Video Data Encoding .....	16-134
Motion Estimation.....	16-135

Quantization .....	16-136
QuantIntra_MPEG2 .....	16-138
Quant_MPEG2 .....	16-139
Huffman Encoding Functions .....	16-140
CreateRLEncodeTable .....	16-140
PutIntraBlock .....	16-141
PutNonIntraBlock .....	16-142
DV .....	16-144
DV Decoding Functions .....	16-150
Variable Length Decoding .....	16-151
InitAllocHuffmanTable_DV .....	16-153
HuffmanDecodeSegment_DV .....	16-154
FreeHuffmanTable_DV .....	16-156
Inverse Quantization .....	16-156
QuantInv_DV .....	16-156
Inverse Discrete Cosine Transformation .....	16-157
DCT2x4x8Inv .....	16-157
DV Encoding Functions.....	16-159
Discrete Cosine Transformation .....	16-160
DCT2x4x8Fw .....	16-160
CountZeros8x8 .....	16-161
DV Color Conversion Functions.....	16-162
YCrCb411ToYCbCr422_5MBDV, YCrCb411ToYCbCr422_ZoomOut2_5MBDV, YCrCb411ToYCbCr422_ZoomOut4_5MBDV, YCrCb411ToYCbCr422_ZoomOut8_5MBDV .....	16-162
YCrCb411ToYCbCr422_EdgeDV, YCrCb411ToYCbCr422_ZoomOut2_EdgeDV, YCrCb411ToYCbCr422_ZoomOut4_EdgeDV, YCrCb411ToYCbCr422_ZoomOut8_EdgeDV .....	16-163
YCrCb420ToYCbCr422_5MBDV, YCrCb420ToYCbCr422_ZoomOut2_5MBDV, YCrCb420ToYCbCr422_ZoomOut4_5MBDV, YCrCb420ToYCbCr422_ZoomOut8_5MBDV .....	16-164
YCrCb422ToYCbCr422_5MBDV,	

---

YCrCb422ToYCbCr422_ZoomOut2_5MBDV, YCrCb422ToYCbCr422_ZoomOut4_5MBDV, YCrCb422ToYCbCr422_ZoomOut8_5MBDV .....	16-166
MPEG-4 .....	16-168
MPEG-4 Video Decoder Functions .....	16-172
High Level Description .....	16-173
Data Types and Structures .....	16-175
Motion Compensation .....	16-184
Copy8x8QP_MPEG4, Copy16x8QP_MPEG4, Copy16x16QP_MPEG4 .....	16-184
OBMC8x8HP_MPEG4, OBMC16x16HP_MPEG4, OBMC8x8QP_MPEG4, .....	16-185
Sprite and Global Motion Compensation .....	16-187
WarpInit_MPEG4 .....	16-187
WarpGetSize_MPEG4 .....	16-188
WarpLuma_MPEG4 .....	16-189
WarpChroma_MPEG4 .....	16-190
CalcGlobalMV_MPEG4 .....	16-191
ChangeSpriteBrightness_MPEG4 .....	16-191
Motion Vector Decoding and Padding .....	16-193
DecodePadMV_PVOP_MPEG4 .....	16-193
DecodeMV_BVOP_Forward_MPEG4 .....	16-195
DecodeMV_BVOP_Backward_MPEG4 .....	16-196
DecodeMV_BVOP_Interpolate_MPEG4 .....	16-197
DecodeMV_BVOP_Direct_MPEG4 .....	16-198
DecodeMV_BVOP_DirectSkip_MPEG4 .....	16-200
LimitMVToRect_MPEG4 .....	16-201
Coefficient Prediction and Reconstruction .....	16-202
PredictReconCoefIntra_MPEG4 .....	16-202
Motion Padding .....	16-204
PadMBHorizontal_MPEG4 .....	16-204
PadMBVertical_MPEG4 .....	16-205
PadMBGray_MPEG4 .....	16-206

PadCurrent_16x16_MPEG4, PadCurrent_8x8_MPEG4 .....	16-207
Vector Padding .....	16-209
PadMV_MPEG4 .....	16-209
Inverse Quantization .....	16-210
QuantInvIntraInit_MPEG4, QuantInvInterInit_MPEG4 .....	16-210
QuantInvIntraGetSize_MPEG4, QuantInvInterGetSize_MPEG4 .....	16-211
QuantInvIntra_MPEG4, QuantInvInter_MPEG4 .....	16-212
VLC Decoding .....	16-214
DecodeDCIntra_MPEG4 .....	16-214
DecodeCoeffsIntra_MPEG4 .....	16-215
DecodeCoeffsIntraRVLCBack_MPEG4 .....	16-216
DecodeCoeffsInter_MPEG4 .....	16-217
DecodeCoeffsInterRVLCBack_MPEG4 .....	16-219
ReconstructCoeffsInter_MPEG4 .....	16-220
DecodeVLCZigzag_IntraDCVLC_MPEG4, DecodeVLCZigzag_IntraACVLC_MPEG4 .....	16-221
DecodeVLCZigzag_Inter_MPEG4 .....	16-223
Block Decoding .....	16-224
DecodeBlockCoef_Intra_MPEG4 .....	16-224
DecodeBlockCoef_Inter_MPEG4 .....	16-226
Postprocessing .....	16-227
FilterDeblocking8x8HorEdge_MPEG4, FilterDeblocking8x8VerEdge_MPEG4 .....	16-227
FilterDeringingThreshold_MPEG4 .....	16-228
FilterDeringingSmooth8x8_MPEG4 .....	16-229
Shape Decoding .....	16-230
DecodeCAEIntraH_MPEG4, DecodeCAEIntraV_MPEG4 .....	16-230
DecodeCAEInterH_MPEG4, DecodeCAEInterV_MPEG4 .....	16-231
DecodeMVS_MPEG4 .....	16-233

PadMBPartial_MPEG4 .....	16-234
PadMBTransparent_MPEG4 .....	16-235
PadMBOpaque_MPEG4 .....	16-237
MPEG-4 Video Encoder Functions .....	16-238
Data Types and Structures .....	16-238
Motion Estimation .....	16-242
SumNorm_VOP_MPEG4 .....	16-242
BlockMatch_Integer_16x16_SEA .....	16-243
MotionEstimation_16x16_SEA .....	16-245
BlockMatch_Integer_16x16_MVFAST .....	16-246
MotionEstimation_16x16_MVFAST .....	16-248
ComputeTextureErrorBlock_SAD .....	16-249
ComputeTextureErrorBlock .....	16-250
Quantization .....	16-251
QuantIntraInit_MPEG4,	
QuantInterInit_MPEG4 .....	16-251
QuantIntraGetSize_MPEG4,	
QuantInterGetSize_MPEG4 .....	16-252
QuantIntra_MPEG4,	
QuantInter_MPEG4 .....	16-253
VLC Encoding .....	16-255
EncodeDCIntra_MPEG4 .....	16-255
EncodeCoeffsIntra_MPEG4 .....	16-256
EncodeCoeffsInter_MPEG4 .....	16-257
EncodeVLCZigzag_IntraDCVLC_MPEG4,	
EncodeVLCZigzag_IntraACVLC_MPEG4 .....	16-258
EncodeVLCZigzag_Inter_MPEG4 .....	16-259
Block Encoding .....	16-260
TransRecBlockCoef_inter_MPEG4 .....	16-260
TransRecBlockCoef_intra_MPEG4 .....	16-261
MV Encoding .....	16-263
FindMVPred_MPEG4 .....	16-263
EncodeMV_MPEG4 .....	16-264

H.261 .....	16-266
H.261 Decoder Functions .....	16-267
Decoding INTRA and INTER Macroblocks .....	16-268
DecodeCoeffsIntra_H261 .....	16-270
DecodeCoeffsInter_H261 .....	16-271
ReconstructCoeffsIntra_H261 .....	16-272
ReconstructCoeffsInter_H261 .....	16-274
H.261 Encoder Functions .....	16-276
EncodeCoeffsIntra_H261 .....	16-277
EncodeCoeffsInter_H261 .....	16-278
Filter8x8_H261 .....	16-279
H.263 .....	16-281
H.263 Decoder Functions .....	16-283
INTRA and INTER Macroblocks Decoding .....	16-284
VLC Decoding .....	16-286
DecodeBlockCoef_Intra_H263 .....	16-286
DecodeBlockCoef_Inter_H263 .....	16-287
DecodeDCIntra_H263 .....	16-288
DecodeCoeffsIntra_H263 .....	16-289
DecodeCoeffsInter_H263 .....	16-291
Inverse Quantization .....	16-292
QuantInvIntra_H263 .....	16-292
QuantInvInter_H263 .....	16-294
Prediction .....	16-296
AddBackPredPB_H263 .....	16-296
Frame Expansion .....	16-297
ExpandFrame_H263 .....	16-297
Resampling .....	16-298
Resample_H263 .....	16-298
UpsampleFour_H263 .....	16-300
DownsampleFour_H263 .....	16-301
UpsampleFour8x8_H263 .....	16-302
SpatialInterpolation_H263 .....	16-303
Boundary Filtering .....	16-305

---

FilterBlockBoundaryHorEdge_H263, FilterBlockBoundaryVerEdge_H263 .....	16-305
FilterDeblocking8x8HorEdge_H263, FilterDeblocking8x8VerEdge_H263 .....	16-306
FilterDeblocking16x16HorEdge_H263, FilterDeblocking16x16VerEdge_H263 .....	16-308
Middle Level Functions .....	16-309
ReconstructCoeffsIntra_H263 .....	16-309
ReconstructCoeffsInter_H263 .....	16-311
H.263 Encoder Functions .....	16-313
VLC Encoding .....	16-314
EncodeDCIntra_H263 .....	16-314
EncodeCoeffsIntra_H263 .....	16-315
EncodeCoeffsInter_H263 .....	16-316
Quantization .....	16-318
QuantIntra_H263 .....	16-318
QuantInter_H263 .....	16-319
Resampling .....	16-320
DownsampleFour16x16_H263 .....	16-320
H.264 .....	16-321
H.264 Decoder Functions .....	16-330
CAVLC Parsing .....	16-335
DecodeCAVLCCoeffs_H264 .....	16-335
DecodeCAVLCChromaDcCoeffs_H264 .....	16-337
DecodeExpGolombOne_H264 .....	16-338
Inverse Quantization and Inverse Transform .....	16-339
TransformDequantLumaDC_H264 .....	16-339
TransformDequantChromaDC_H264 .....	16-340
DequantTransformResidual_H264 .....	16-341
DequantTransformResidualAndAdd_H264 .....	16-343
TransformPrediction_H264 .....	16-345
DequantTransformResidual_SISP_H264 .....	16-346
TransformDequantChromaDC_SISP_H264 .....	16-347
Intra Prediction .....	16-348



PredictIntra_4x4_H264 .....	16-348
PredictIntra_16x16_H264 .....	16-350
PredictIntraChroma8x8_H264 .....	16-351
Inter Prediction .....	16-353
ExpandPlane_H264 .....	16-353
InterpolateLuma_H264 .....	16-354
InterpolateLumaTop_H264 .....	16-356
InterpolateLumaBottom_H264 .....	16-358
InterpolateChroma_H264 .....	16-361
InterpolateChromaTop_H264 .....	16-363
InterpolateChromaBottom_H264 .....	16-365
InterpolateBlock_H264 .....	16-367
WeightedAverage_H264 .....	16-368
UniDirWeightBlock_H264 .....	16-369
BiDirWeightBlock_H264 .....	16-371
BiDirWeightBlockImplicit_H264 .....	16-372
Macroblock Reconstruction .....	16-374
ReconstructChromaInterMB_H264 .....	16-374
ReconstructChromaIntraHalvesMB_H264 .....	16-376
ReconstructChromaIntraMB_H264 .....	16-378
ReconstructChromaInter4x4MB_H264 .....	16-380
ReconstructChromaIntraHalves4x4MB_H264 .....	16-382
ReconstructChromaIntra4x4MB_H264 .....	16-384
ReconstructLumaInterMB_H264 .....	16-386
ReconstructLumaIntraHalfMB_H264 .....	16-387
ReconstructLumaIntraMB_H264 .....	16-389
ReconstructLumaInter4x4MB_H264 .....	16-391
ReconstructLumaIntraHalf4x4MB_H264 .....	16-392
ReconstructLumaIntra4x4MB_H264 .....	16-394
ReconstructLumaInter8x8MB_H264 .....	16-395
ReconstructLumaIntraHalf8x8MB_H264 .....	16-397
ReconstructLumaIntra8x8MB_H264 .....	16-398
ReconstructLumaIntra16x16MB_H264 .....	16-400
ReconstructLumaIntra_16x16MB_H264 .....	16-402

Deblocking Filtering .....	16-404
FilterDeblockingLuma_VerEdge_H264 .....	16-404
FilterDeblockingLuma_VerEdge_MBAFF_H264 .....	16-406
FilterDeblockingLuma_HorEdge_H264 .....	16-407
FilterDeblockingChroma_VerEdge_H264 .....	16-409
FilterDeblockingChroma_VerEdge_MBAFF_H264 .....	16-412
FilterDeblockingChroma_HorEdge_H264 .....	16-413
H.264 Encoder Functions.....	16-416
Forward Transform and Quantization .....	16-417
TransformQuantChromaDC_H264 .....	16-417
TransformQuantLumaDC_H264 .....	16-419
TransformQuantResidual_H264 .....	16-421
TransformLuma8x8Fwd_H264 .....	16-422
QuantLuma8x8_H264 .....	16-423
GenScaleLevel8x8_H264 .....	16-424
CAVLC Functions .....	16-425
EncodeCoeffsCAVLC_H264 .....	16-425
EncodeChromaDcCoeffsCAVLC_H264 .....	16-429
Inverse Quantization and Transform.....	16-430
QuantLuma8x8Inv_H264 .....	16-430
TransformLuma8x8InvAddPred_H264 .....	16-431

## Appendix A Handling of Special Cases

## Appendix B Interpolation in Image Geometric Transform Functions

Overview of Interpolation Modes .....	B-1
Mathematical Notation .....	B-3
Nearest Neighbor Interpolation.....	B-3
Linear Interpolation .....	B-3
Cubic Interpolation.....	B-4
Super Sampling .....	B-6
Lanczos Interpolation.....	B-7

## **Appendix C Removed Functions**

### **Bibliography**

### **Glossary**

### **Index**

# Overview

---

# 1

This manual describes the structure, operation, and functions of the Intel® Integrated Performance Primitives (Intel® IPP) for Intel® architecture that operate on two-dimensional signals and are used for image and video processing. This is the second volume of the Intel IPP Reference Manual, which also comprises descriptions of the Intel IPP for signal processing (volume 1), operations on small matrices (volume 3), and cryptography functions (volume 4).

The Intel IPP software package supports many functions whose performance can be significantly enhanced on Intel architecture, particularly using the MMX™ technology, Streaming SIMD Extensions (SSE), Streaming SIMD Extensions 2 (SSE2), Streaming SIMD Extensions 3 (SSE3), as well as Intel® Itanium® architecture. For information on Intel IPP implementation for Intel® PCA application processors, see [Cross Architecture Alignment](#) section later in this chapter.

The Intel IPP for image and video processing software is a collection of low-overhead, high-performance operations performed on two-dimensional (2D) arrays of pixels.

This manual explains the Intel IPP concepts as well as specific data type definitions and operation models used in the image and video processing domain and provides detailed descriptions of the Intel IPP image and video processing functions.

This chapter introduces the Intel IPP software and explains the organization of this manual.

## About This Software

Intel IPP software enables taking advantage of the parallelism of the single-instruction, multiple data (SIMD) instructions that comprise the core of the MMX technology and Streaming SIMD Extensions. These technologies improve the performance of computation-intensive signal, image, and video processing applications. Use of Intel IPP primitive functions can help to drastically reduce development costs and accelerate time-to-market by eliminating the need of writing processor-specific code for computation intensive routines.

## Hardware and Software Requirements

Intel IPP for Intel architecture software runs on personal computers that are based on IA-32 processors or Itanium® architecture-based processors and running Microsoft® Windows® 2000, Windows® ME, Windows® XP, or Linux®. Intel IPP integrates into the customer's application or library written in C or C++.

## Platforms Supported

Intel IPP for Intel architecture software runs on Windows and Linux platforms. The code and syntax used in this manual for function and variable declarations are written in the ANSI C style. However, versions of Intel IPP for different processors or operating systems may, of necessity, vary slightly.

## Cross Architecture Alignment

### Cross Architecture Overview

Intel IPP has been designed to support application development on various Intel® architectures. Previously, Intel IPP was offered in two separate products, one for the Intel® Pentium®, Intel® Xeon™ and Intel® Itanium® processors and one for the Intel® PCA processors based on Intel XScale® technology. While these separate packages included many similarities, prior to version 4.0, there were some differences in both functionality and interface.

With rising interest in cross architecture development, the Intel IPP development teams have worked to bridge these differences to allow for easy development of applications for multiple Intel® platforms. Starting from version 4.0 onwards, Intel IPP represents the result of this effort and provides alignment of the two separate packages into a single offering that includes support for all of these architectures. This includes interface alignment and full API alignment for all functions that are relevant to both architectures.

With this release, the functions available for Intel Pentium, Xeon and Itanium processors represent a superset of functions available for the Intel PCA processors. This means the API definition is common for all processors, while the underlying function implementation takes into account the variations in processor architectures.

By providing a single cross-architecture API, Intel IPP allows software application repurposing and enables developers to port to unique features across Intel® processor-based desktop, server, mobile, and handheld platforms. Developers can write their code once in order to realize the application performance over many processor generations.

For additional information on API additions and changes from previous versions, please refer to the product release notes and the document *Migration Guide for Intel® IPP Cross Architecture API Alignment* available at <http://www.intel.com/software/products/ipp>.

The following table summarizes the functionality covered in each Intel IPP implementation.

**Table 1-1 Function Coverage in Intel IPP**

Function Group	Intel® Pentium® 4 processors	Intel® Xeon™ processors with Intel® EM64T	Intel® Itanium® 2 processors	Intel® PCA processors	Intel® IXP4XX Product Line
Signal Processing	available	available	available	available <sup>1</sup>	available
Image Processing	available	available	available	available <sup>1</sup>	available
JPEG	available	available	available	available <sup>1</sup>	available
Speech Recognition	available	available	available	n/a	available
Speech Coding	available	available	available	available <sup>1</sup>	available
Audio Codecs	available	available	available	available <sup>1</sup>	available
Video Codecs	available	available	available	available <sup>1</sup>	available
Matrix	available	available	available	n/a	n/a
Vector Math	available	available	available	n/a	n/a
Computer Vision	available	available	available	n/a	available
Cryptography	available	available	available	available	available
Data Compression	available	available	available	n/a	available
Color Conversion	available	available	available	n/a	available
String Processing	available	available	available	n/a	available

1. Intel PCA processors support a subset of these functions.

## API Changes in Version 5.0

Starting from the version 5.0 Intel IPP has a limited compatibility with the previous versions. To improve the library in several aspects, the following changes have been made:

- Some functions have been replaced by new functions with extended functionality.
- Some functions have been changed to make API more consistent and handy.
- Odd and unusable functions have been discarded.

- Several complicated functions have been discarded from the Intel IPP, but their functionalities have been moved to the codec/sample level.

All these changes affect existent applications. [Table 1](#) in the [Appendix C](#) lists such functions for the image processing domains and specifies the corresponding Intel IPP 5.0 functions to replace them. If an application calls functions listed in this table, then the source code should be modified.

Several functions, for example, all color conversion functions, have been moved to the newly created domains. These changes do not affect existent applications.

Additionally, certain modifications breaking binary compatibility have been made in the Intel IPP 5.0, for example, directory tree structure has been modified, the `ipp20` folder has been discarded. These changes also affect existent applications that should be at least rebuilt.

Version 5.1, as well as the subsequent versions, has full backward compatibility with version 5.0.

## Technical Support

Intel IPP provides a product web site that offers timely and comprehensive product information, including product features, white papers, and technical articles. For the latest information, see: <http://developer.intel.com/software/products/>.

Intel also provides a support web site that contains a rich repository of self-help information, including getting started tips, known product issues, product errata, license information, and more (visit <http://support.intel.com/support/>).

Registering your product entitles you to one-year technical support and product updates through Intel® Premier Support. Intel Premier Support is an interactive issue management and communication web site providing the following services:

- Submit issues and review their status.
- Download product updates anytime of the day.

To register your product, or contact Intel, or seek product support, please visit: <http://www.intel.com/software/products/support>

## Intel® IPP Code Samples

An extensive library of code samples and codecs has been implemented using the Intel IPP functions to demonstrate the use of Intel IPP and to help accelerate the development of your application, components, and codecs. The samples can be downloaded from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>.

## About This Manual

This manual provides a background for the image and video processing concepts used in the Intel IPP software as well as detailed description of the respective Intel IPP functions. The Intel IPP functions are combined in groups by their functionality. Each group of functions is described in a separate chapter (chapters 3 through 16).

## Manual Organization

This manual contains the following chapters:

- Chapter 1 “[Overview](#).” Introduces the Intel IPP software for image and video processing, provides information on manual organization, and explains notational conventions.
- Chapter 2 “[Intel Integrated Performance Primitives Concepts](#).” Explains the basic concepts underlying image processing in Intel IPP and describes the supported data formats and operation modes.
- Chapter 3 “[Support Functions](#).” Describes functions that are used to support the operation of Intel IPP software, such as memory allocation functions and status information function.
- Chapter 4 “[Image Data Exchange and Initialization Functions](#).” Describes image processing primitive functions that are used to set, copy, convert, and scale images, as well as initialize specific images.
- Chapter 5 “[Image Arithmetic and Logical Operations](#).” Describes Intel IPP image processing functions that modify pixel values using arithmetic, logical, and alpha composition operations.
- Chapter 6 “[Image Color Conversion](#).” Describes the Intel IPP color space conversion operations.
- Chapter 7 “[Threshold and Compare Operations](#).” Describes functions that perform thresholding and image comparison operations.
- Chapter 8 “[Morphological Operations](#).” Describes common and advanced morphological operations as well as operations with gray kernel and morphological reconstruction.
- Chapter 9 “[Filtering Functions](#).” Describes Intel IPP filtering operations using linear and non-linear filters.
- Chapter 10 “[Image Linear Transforms](#).” Describes the Fast Fourier Transform (FFT), Discrete Fourier Transform (DFT), and Discrete Cosine Transform (DCT) functions implemented in the Intel IPP for image processing.



- Chapter 11 “[Image Statistics Functions](#).” Describes Intel IPP functions that compute image statistical properties such as image norms and moments.
- Chapter 12 “[Image Geometric Transforms](#).” Describes the supported geometric transformations of images.
- Chapter 13 “[Wavelet Transforms](#).” Describes the supported functions for wavelet decomposition and reconstruction of images.
- Chapter 14 “[Computer Vision](#).” Describes Intel IPP functions that are specific for computer vision applications.
- Chapter 15 “[Image Compression Functions](#).” Describes Intel IPP functions that perform still image compression and coding in accordance with JPEG and JPEG2000 standards.
- Chapter 16 “[Video Coding](#).” Describes Intel IPP functions that support encoding and decoding of video data according to MPEG-1, MPEG-2 and MPEG-4 standards, as well as functions that support H.263, H.264, and DV specifications.

The manual also includes a [Glossary](#) of terms, a [Bibliography](#), and an [Index](#), as well as [Appendix A](#) that describes handling of special cases by Intel IPP functions, [Appendix B](#) that describes the interpolation algorithms used in the geometric transformation functions of the Intel IPP, and [Appendix C](#) that lists functions removed from Intel IPP version 5.0.

## Function Descriptions

In Chapters 3 through 16, each function is introduced by its short name (without the `ippi` prefix and descriptors) and a brief description of its purpose. This is followed by the function call sequence, definitions of all function arguments, and a more detailed explanation of the function purpose. The following sections are included in the function descriptions:

<i>Syntax</i>	Lists function prototypes.
<i>Parameters</i>	Describes all function parameters.
<i>Description</i>	Defines the function and details the operation performed by the function. Code examples and equations that the function implements may be included in the description.
<i>Return Values</i>	Describes values indicating status codes set as the result of the function execution.

## Audience for This Manual

The manual is intended for the developers of image and video processing applications and libraries, as well as cross-domain applications. The audience must have experience in using C and working knowledge of the vocabulary and principles of image and video processing.

## Online Version

This manual is available in an electronic format (Portable Document Format, or PDF). To obtain a hard copy of the manual, print the file using the printing capability of Adobe Acrobat\*, the tool used for the online presentation of the document.

## Related Publications

For more information about image and video processing concepts and algorithms, refer to the books and materials listed in the [Bibliography](#).

## Notational Conventions

In this manual, notational conventions include:

- Fonts used for distinction between the text and the code
- Naming conventions for different items.

## Font Conventions

The following font conventions are used in this manual:

THIS TYPE STYLE	Used in the text for Intel IPP constant identifiers; for example, <code>IPPI_INTER_LINEAR</code> .
This type style	Mixed with the uppercase in structure names as in <code>IppiSize</code> ; also used in function names, code examples and call statements; for example, <code>ippiMomentInitAlloc()</code> .
<i>This type style</i>	Parameters in function prototypes and parameters description; for example: <i>value</i> , <i>srcStep</i> .

## Naming Conventions

The following naming conventions for different items are used by the Intel IPP software:

- Constant identifiers are in uppercase; for example, `IPPI_INTER_CUBIC`.

- All structures and enumerators, specific for the image and video processing domain have the `Ippi` prefix, while those common for entire Intel IPP software have the `Ipp` prefix; for example, `IppiPoint`, `IppDitherType`.
- All names of the functions used for image and video processing have the `ippi` prefix. In code examples, you can distinguish the Intel IPP interface functions from the application functions by this prefix.



---

**NOTE.** In this manual, the `ippi` prefix in function names is always used in code examples and function prototypes. In the text, this prefix is usually omitted when referring to the function group.

---

- Each new part of a function name starts with an uppercase character, without underscore; for example, `ippiGetSpatialMoment`.

For the detailed description of function name structure in Intel IPP, see [Function Naming](#) in Chapter 2.

# *Intel® Integrated Performance Primitives Concepts*

---

## 2

This chapter explains the purpose and structure of the Intel® Integrated Performance Primitives (Intel® IPP) for Intel® Architecture software and looks over some of the basic concepts used in the image and video processing part of Intel IPP. It also describes the supported data formats and operation modes, and defines function naming conventions in the manual.

## **Basic Features**

The Intel Integrated Performance Primitives, like other members of the Intel® Performance Libraries, is a collection of high-performance code that performs domain-specific operations. It is distinguished by providing a low-level, stateless interface.

Based on experience in developing and using Intel Performance Libraries, Intel IPP has the following major distinctive features:

- The Intel IPP provides basic low-level functions for creating applications in several different domains, such as signal processing, image and video processing, and operations on small matrices;
- Intel IPP functions follow the same interface conventions including uniform naming rules and similar composition of prototypes for primitives that refer to different application domains;
- Intel IPP functions use abstraction level which is best suited to achieve superior performance figures by the application programs.

To speed up performance, Intel IPP functions are optimized to use all benefits of Intel® architecture processors. Besides that, most of Intel IPP functions do not use complicated data structures, which helps reduce overall execution overhead.

The Intel IPP software work with two-dimensional arrays of data, not an image abstraction, thus providing lower-level image processing functions than in most other image processing libraries. This serves the following major goals:

- To free developers from the necessity of filling specific structures that describe image data and architecture;
- To achieve a maximum function performance by avoiding the structures' field parsing and code branching;
- To provide a collection of simple and obvious lower-level functions to facilitate incorporation within an existing application or architecture.

Intel IPP is well-suited for cross-platform applications. For example, the functions developed for IA-32 platform can be readily ported to Intel® Itanium®-based platforms and systems with Intel® StrongARM\* technology or Intel XScale® technology.

For more information on platform compatibility, see [Cross Architecture Alignment](#) section in chapter 1.

## Function Naming

Naming conventions for the Intel IPP functions are similar for all covered domains. You can distinguish signal processing functions by the `ipps` prefix, while image and video processing functions have `ippi` prefix, and functions that are specific for operations on small matrices have `ippm` prefix in their names.

Function names in Intel IPP have the following general format:

```
ipp<data-domain><name>_<datatype>[_<descriptor>](<arguments>);
```

The elements of this format are explained in the sections that follow.

## Data-Domain

The *data-domain* is a single character that expresses the subset of functionality to which a given function belongs. The current version of Intel IPP supports the following data-domains:

- |   |  |
|---|--|
| s | for signals (expected data type is a 1D signal);         |
| i | for images and video (expected data type is a 2D image); |
| m | for matrices (expected data type is a matrix).           |

For example, function names that begin with `ippi` signify that respective functions are used for image or video processing.

## Name

The *name* is an abbreviation for the core operation that the function really does, for example `Add`, `Sqrt`, followed in some cases by a function-specific modifier:

`<name> = <operation>[_modifier]`

This modifier, if present, denotes a slight modification or variation of the given function.

## Data Types

The *datatype* field indicates data types used by the function, in the following format:

`<bit depth><bit interpretation> ,`

where

`bit depth = <1|8|16|32|64>`

and

`bit interpretation = <u|s|f>[c]`

Here *u* indicates “unsigned integer”, *s* indicates “signed integer”, *f* indicates “floating point”, and *c* indicates “complex”.

The Intel IPP supports the following data types for image and video processing functions:

- `8u`            8-bit, unsigned data
- `8s`            8-bit, signed data
- `16u`           16-bit, unsigned data
- `16s`           16-bit, signed data
- `16sc`          16-bit, complex short data
- `32u`           32-bit, unsigned data
- `32s`           32-bit, signed data
- `32sc`          32-bit, complex int data
- `32f`           32-bit, single-precision real floating point data
- `32fc`          32-bit, single-precision complex floating point data

- 64s 64-bit, quadword signed data
- 64f 64-bit, double-precision real floating point data




---

**NOTE.** Intel IPP does not support `1u` data type, therefore bitonal images should be converted to `8u` gray scale images for further processing by the library functions. There is a special function for such conversion `ippiConvert_1u8u_C1R..`

---

The formats for complex data are represented in Intel IPP by structures defined as follows:

```
typedef struct {
    Ipp16s re;
    Ipp16s im;
} Ipp16sc;

typedef struct {
    Ipp32s re;
    Ipp32s im;
} Ipp32sc;

typedef struct {
    Ipp32f re;
    Ipp32f im;
} Ipp32fc;
```

where `re`, `im` denote the real and imaginary part, respectively.

Complex data formats are used by several arithmetic image processing functions. The `32fc` format is also used to store input/output data in some Fourier transform functions. The 64-bit formats, `64s` and `64f`, are used for storing data computed by some image statistics functions.

For functions that operate on a single data type, the `datatype` field contains only one of the values listed above.

If a function operates on source and destination images that have different data types, the respective data type identifiers are listed in the function name in order of source and destination as follows:

$$\langle datatype \rangle = \langle src1Depth \rangle [src2Depth] [dstDepth]$$

For example, the function that converts 8-bit unsigned source image data to 32-bit floating point destination image data has the `8u32f` value for the `datatype` field.



**NOTE.** In the lists of function parameters (arguments), the `Ipp` prefix is written in the data type. For example, the 8-bit unsigned data is denoted as `Ipp8u` type. These Intel IPP -specific data types are defined in the respective library header files.

## Descriptor

The *descriptor* field further describes the data associated with the operation. It may contain implied parameters and/or indicate additional required parameters.

To minimize the number of code branches in the function and thus reduce potentially unnecessary execution overhead, most of the general functions are split into separate primitive functions, with some of their parameters entering the primitive function name as descriptors.

However, where the number of permutations of the function becomes large and unreasonable, some functions may still have arguments that determine internal operation (for example, `ippiThreshold`).

The following descriptors are used in image and video processing functions:

- **A** Data contains an alpha channel (always the last channel, requires C4, not processed)
- **Cn** Data is made up of *n* discrete interleaved channels (1, 2, 3, 4)
- **C** Channel of interest (COI) is used in the operation
- **D2** Image is two-dimensional
- **I** Operation is in-place
- **M** Uses mask ROI for source and destination images
- **Pn** Data is made up of *n* discrete planar (non-interleaved) channels, with a separate pointer to each plane
- **R** Uses region of interest (ROI)
- **Sfs** Saturation and fixed scaling mode is used

The abbreviations of descriptors in function names are always presented in alphabetical order.



Not all functions have every abbreviation listed above. For example, in-place mode makes no sense for a copy operation.

Also, some data descriptors are implied when dealing with some operations.

The default for image processing functions is to operate on a two-dimensional image and to saturate the results without scaling them. In these cases, the implied abbreviations `D2` and `Sns` (saturation and no scaling) are not contained in the function name.

## Arguments

The arguments in functions are in the following order: all source operands; all destination operands; all other, operation-specific arguments.

Source arguments are named "`Src`" or "`SrcN`," if there is more than one input image. Destination arguments are named "`Dst`". For in-place operations, the input/output argument contains the name "`SrcDst`." All arguments defined as pointers start with lowercase `p`, for example, `pSrc`, `pMean`, `pSpec`.

## Function Prototypes in Intel IPP

Function names in Intel IPP contain *datatype* and *descriptor* fields after the *name* field (see [Function Naming](#) in this chapter). Most Intel IPP functions for image processing have a number of flavors that differ in data types associated with the operation, and in some additional parameters.

Each function flavor has its unique prototype used in function definition and for calling the function from the application program. For many flavors of a given function, these prototypes look quite similar.

To avoid listing all the similar prototypes in function description sections of some chapters in this manual, only different templates for such prototypes followed by the table of applicable data types and descriptors for each function may be given. For simplicity, in such cases the data type and descriptor fields in the function name are denoted as *mod*:

`<mod> = <datatype>_<descriptor>`

For example, the template for the prototype of the image dilation function that performs not-in-place operation, looks like this:

```
ippStatus ippiDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

where the supported values for *mod* are:

```
8u_C1R
8u_C3R
8u_AC4R
```

This notation means that the `ippiDilate` function has three flavors for a not-in-place operation, which process 8-bit unsigned data (of `Ipp8u` type) and differ in the number of channels in processed images. These flavors have the following prototypes:

```
IppStatus ippiDilate_8u_C1R(const Ipp8u* pSrc, int srcStep,
                             Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDilate_8u_C3R(const Ipp8u* pSrc, int srcStep,
                             Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDilate_8u_AC4R(const Ipp8u* pSrc, int srcStep,
                             Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Thus, to obtain the full name and arguments list for the specific function flavor, not listed directly, do the following:

- choose the function operation mode (denoted in this manual as **Case 1, 2,...**) and look in the table for the supported data types and descriptors;
- set the *mod* field in the function name as the concatenation of chosen data type and descriptor, delimited by the underscore;
- use the respective template, substituting all the *datatype* fields in the arguments list with the chosen data type. Note that `Ipp` prefix is written before the *datatype* in the arguments list (see [Data Types](#) in this chapter for details).

*Example.*

To get the prototype for the [ippiSet](#) function flavor that sets each channel of a 3-channel destination image to 16-bit signed values, choose **Case 2: Setting each color channel to a specified value** and use *datatype* = `16s`, *descriptor* = `C3R`.

After substituting the *mod* field with `16s_C3R`, obtain the required prototype as

```
ippiSet_16s_C3R(const Ipp16s value[3], Ipp16s* pDst, int dstStep,
                IppiSize roiSize);
```

## Integer Result Scaling

Some image processing functions operating on integer data use scaling of the internally computed output results by the integer *scaleFactor*, which is specified as one of the function parameters. These functions have the *Sfs* descriptor in their names.

The scale factor can be negative, positive, or zero. Scaling is applied because internal computations are generally performed with a higher precision than the data types used for input and output images.



---

**NOTE.** The result of integer operations is always saturated to the destination data type range even when scaling is used.

---

The scaling of an integer result is done by multiplying the output pixel values by  $2^{-scaleFactor}$  before the function returns. This helps retain either the output data range or its precision. Usually the scaling with a positive factor is performed by the shift operation. The result is rounded off to the nearest integer number.

For example, the integer *Ipp16s* result of the square operation *ippiSqr* for the input value 200 is equal to 32767 instead of 40000, that is, the result is saturated and the exact value can not be restored. The scaling of the output value with the factor *scaleFactor* = 1 yields the result 20000 which is not saturated, and the exact value can be restored as  $20000 * 2$ . Thus, the output data range is retained.

The following example shows how the precision can be partially retained by means of scaling. The integer square root operation *ippiSqrt* (without scaling) for the input value 2 gives the result equal to 1 instead of 1.414. Scaling of the internally computed output value with the factor *scaleFactor* = -3 gives the result 11, and permits to restore the more precise value as  $11 * 2^{-3} = 1.375$ .

## Error Reporting

Intel IPP functions return status codes of the performed operation to report errors and warnings to the calling program. Thus, it is up to the application to perform error-related actions and/or recover from the error. The last value of the error status is not stored, and the user is to decide whether to check it or not as the function returns.

The status codes are of *IppStatus* type and are global constant integers.

The status codes and corresponding messages reported by the Intel IPP for image and video processing are listed in [Table 2-1](#).

**Table 2-1 Status Codes and Messages**

Status Code	Message
ippStsNotSupportedModeErr	The requested mode is currently not supported
ippStsResizeNoOperationErr	One of the output image dimensions is less than 1 pixel
ippStsBlockStepErr	Step for Block less than 8
ippStsMBStepErr	Step for MB less than 16
ippStsNoiseRangeErr	Noise value for Wiener Filter is out of range
ippStsLPCCalcErr	Linear prediction could not be evaluated
ippStsJPEG2KBadPassNumber	Pass number exceeds allowed limits [0,nOfPasses-1]"
ippStsJPEG2KDamagedCodeBlock	Codeblock for decoding is damaged
ippStsH263CBPYCodeErr	Illegal Huffman code during CBPY stream processing
ippStsH263MCBPCInterCodeErr	Illegal Huffman code while MCBPC Inter stream processing
ippStsH263MCBPCIntraCodeErr	Illegal Huffman code while MCBPC Intra stream processing
ippStsNotEvenStepErr	Step value is not pixel multiple
ippStsHistoNofLevelsErr	Number of levels for histogram is less than 2
ippStsLUTNofLevelsErr	Number of levels for LUT is less than 2
ippStsMP4BitOffsetErr	Incorrect bit offset value
ippStsMP4QPErr	Incorrect quantization parameter
ippStsMP4BlockIdxErr	Incorrect block index
ippStsMP4BlockTypeErr	Incorrect block type
ippStsMP4MVCodeErr	Illegal Huffman code while MV stream processing
ippStsMP4VLCCodeErr	Illegal Huffman code while VLC stream processing
ippStsMP4DCCodeErr	Illegal code while DC stream processing
ippStsMP4FcodeErr	Incorrect fcode value
ippStsMP4AlignErr	Incorrect buffer alignment
ippStsMP4TempDiffErr	Incorrect temporal difference
ippStsMP4BlockSizeErr	Incorrect size of block or macroblock
ippStsMP4ZeroBABErr	All BAB values are zero
ippStsMP4PredDirErr	Incorrect prediction direction
ippStsMP4BitsPerPixelErr	Incorrect number of bits per pixel
ippStsMP4VideoCompModeErr	Incorrect video component mode

**Table 2-1 Status Codes and Messages (continued)**

ippStsMP4LinearModeErr	Incorrect DC linear mode
ippStsH263PredModeErr	Prediction Mode value error
ippStsH263BlockStepErr	The step value is less than 8
ippStsH263MBStepErr	The step value is less than 16
ippStsH263FrameWidthErr	The frame width is less than 8
ippStsH263FrameHeightErr	The frame height is less than or equal to zero
ippStsH263ExpandPelsErr	The expand pixels number is less than 8
ippStsH263PlaneStepErr	Step value is less than the plane width
ippStsH263QuantErr	Quantizer value is less than or equal to zero, or greater than 31
ippStsH263MVCodeErr	Illegal Huffman code while MV stream processing
ippStsH263VLCCodeErr	Illegal Huffman code while VLC stream processing
ippStsH263DCCodeErr	Illegal code while DC stream processing
ippStsH263ZigzagLenErr	Zigzag compact length is more than 64
ippStsJPEGHuffTableErr	JPEG Huffman table is destroyed
ippStsJPEGDCTRangeErr	JPEG DCT coefficient is outs of the range
ippStsJPEGOutOfBufErr	Attempt to access out of the buffer
ippStsChannelOrderErr	Wrong order of the destination channels
ippStsZeroMaskValueErr	All values of the mask are zero
ippStsRangeErr	Bad values of bounds: the lower bound is greater than the upper bound
ippStsQPErr	Incorrect value of quantizer parameter
ippStsQuadErr	The quadrangle is nonconvex or degenerates into triangle, line or point
ippStsRectErr	Size of the rectangular region is less than or equal to 1
ippStsCoeffErr	Unallowable values of the transformation coefficients
ippStsNoiseValErr	Bad value of noise amplitude for dithering
ippStsDitherLevelsErr	Number of dithering levels is out of range
ippStsNumChannelsErr	Bad or unsupported number of channels
ippStsDataTypeErr	Bad or unsupported data type
ippStsCOIErr	COI is out of range
ippStsOutOfRangeErr	Argument is out of range or point is outside the image
ippStsDivisorErr	Divisor is equal to zero, function is aborted
ippStsAlphaTypeErr	Illegal type of image compositing operation

**Table 2-1      Status Codes and Messages (continued)**

ippStsGammaRangeErr	Gamma range bound is less than or equal to zero
ippStsGrayCoefSumErr	Sum of the conversion coefficients must be less than or equal to 1
ippStsChannelErr	Illegal channel number
ippStsJaehneErr	Magnitude value is negative
ippStsStepErr	Step value is less than or equal to zero
ippStsStrideErr	Stride value is less than the row length
ippStsEpsValErr	Negative epsilon value error
ippStsScaleRangeErr	Scale bounds are out of the range
ippStsThresholdErr	Invalid threshold bounds
ippStsWtOffsetErr	Invalid offset value of wavelet filter
ippStsAnchorErr	Anchor point is outside the mask
ippStsMaskSizeErr	Invalid mask size
ippStsShiftErr	Shift value is less than zero
ippStsSampleFactorErr	Sampling factor is less than or equal to zero
ippStsResizeFactorErr	Resize factor(s) is less or equal to zero
ippStsDivByZeroErr	An attempt to divide by zero
ippStsInterpolationErr	Invalid interpolation mode
ippStsMirrorFlipErr	Invalid flip mode
ippStsMoment00ZeroErr	Moment value M(0,0) is too small to continue calculations
ippStsThreshNegLevelErr	Negative value of the level in the threshold operation
ippStsContextMatchErr	Context parameter doesn't match the operation
ippStsFftFlagErr	Invalid value of the FFT flag parameter
ippStsFftOrderErr	Invalid value of the FFT order parameter
ippStsMemAllocErr	Not enough memory for the operation
ippStsNullPtrErr	Null pointer error
ippStsSizeErr	Wrong value of data size
ippStsNoErr	No error, it's OK
ippStsNoOperation	No operation has been executed
ippStsMisalignedBuf	Misaligned pointer in operation in which it must be aligned
ippStsSqrtNegArg	Negative value(s) of the argument in the function Sqrt
ippStsInvZero	INF result. Zero value was met by InvThresh with zero level

**Table 2-1**      **Status Codes and Messages (continued)**

<code>ippStsEvenMedianMaskSize</code>	Even size of the Median Filter mask was replaced by the odd number
<code>ippStsDivByZero</code>	Zero value(s) of the divisor in the function <code>Div</code>
<code>ippStsLnZeroArg</code>	Zero value(s) of the argument in the function <code>Ln</code>
<code>ippStsLnNegArg</code>	Negative value(s) of the argument in the function <code>Ln</code>
<code>ippStsNanArg</code>	Not a Number argument value warning
<code>ippStsBadArgErr</code>	Invalid or bad argument
<code>ippStsJPEGMarker</code>	JPEG marker was met in the bitstream
<code>ippStsResFloor</code>	All result values are floored
<code>ippStsAffineQuadChanged</code>	4th vertex of destination quad is not equal to customer's one
<code>ippStsWrongIntersectROI</code>	Wrong ROI that has no intersection with the source or destination image. No operation
<code>ippStsWrongIntersectQuad</code>	Wrong quadrangle that has no intersection with the source or destination ROI. No operation
<code>ippStsSymKernelExpected</code>	The kernel is not symmetric

The status codes ending with `Err` (except for the `ippStsNoErr` status) indicate an error; the integer values of these codes are negative. When an error occurs, the function execution is interrupted.

The status code `ippStsNoErr` indicates no error.

All other status codes indicate warnings. When a specific case is encountered, the function execution will be completed and the corresponding warning status code will be returned. For example, if the function `ippiDiv` meets an attempt to divide a positive value by zero, the function execution is not aborted. The result of the operation is set to the maximum value that can be represented by the source data type, and the user is warned by the output status `ippStsDivByZero`. See appendix A [Handling of Special Cases](#) for more information.

## Structures and Enumerators

This section describes the structures and enumerators used by the Intel Integrated Performance Primitives for image and video processing.

The `IppStatus` constant enumerates the status code values returned by Intel IPP functions, indicating whether the operation was error-free or not.

See section [Error Reporting](#) in this chapter for more information on the set of valid status codes and corresponding error messages for image and video processing functions.

The structure `IppiPoint` for storing the geometric position of a point is defined as

```
typedef struct {  
    int x;  
    int y;  
} IppiPoint;
```

where `x, y` denote the coordinates of the point.

The structure `IppiSize` for storing the size of a rectangle is defined as

```
typedef struct {  
    int width;  
    int height;  
} IppiSize;
```

where `width` and `height` denote the dimensions of the rectangle in the `x`- and `y`- directions, respectively.

The structure `IppiRect` for storing the geometric position and size of a rectangle is defined as

```
typedef struct {  
    int x;  
    int y;  
    int width;  
    int height;  
} IppiRect;
```

where `x, y` denote the coordinates of the top left corner of the rectangle that has dimensions `width` in the `x`-direction by `height` in the `y`-direction.



The `IppiConnectedComp` structure used in [computer vision functions](#) defines the connected component as follows:

```
typedef struct {
    Ipp32s area;
    Ipp32f value;
    IppiRect rect;
} IppiConnectedComp;
```

where `area`, `value`, and `rect` denote parameters of the connected component.

The `IppiMaskSize` enumeration defines the neighborhood area for some [morphological](#) and [filtering functions](#):

```
typedef enum {
    ippMskSize1x3 = 13,
    ippMskSize1x5 = 15,
    ippMskSize3x1 = 31,
    ippMskSize3x3 = 33,
    ippMskSize5x1 = 51,
    ippMskSize5x5 = 55
} IppiMaskSize;
```

The `IppCmpOp` enumeration defines the type of compare operation to be used in image [comparison functions](#):

```
typedef enum {
    ippCmpLess,
    ippCmpLessEq,
    ippCmpEq,
    ippCmpGreaterEq,
    ippCmpGreater
} IppCmpOp;
```

The `IppRoundMode` enumeration defines the rounding mode to be used in [conversion functions](#):

```
typedef enum {
    ippRndZero,
    ippRndNear
} IppRoundMode;
```

The `IppHintAlgorithm` enumeration defines the type of code to be used in some image transform and statistics functions, i. e. faster but less accurate, or vice-versa, more accurate but slower. For more information on using this enumerator, see [Table 10-2](#) or [Table 11-3](#).

```
typedef enum {
    ippAlgHintNone,
    ippAlgHintFast,
    ippAlgHintAccurate
} IppHintAlgorithm;
```

The types of interpolation used by [geometric transform functions](#) are defined as follows:

```
enum {
    IPPI_INTER_NN          = 1,
    IPPI_INTER_LINEAR      = 2,
    IPPI_INTER_CUBIC       = 4,
    IPPI_INTER_SUPER       = 8,
    IPPI_INTER_LANCZOS     = 16,
    IPPI_SMOOTH_EDGE       = (1 << 31)
};
```

The `IppiAlphaType` enumeration defines the type of the compositing operation to be used in the [alpha composition functions](#):

```
typedef enum {
    ippAlphaOver,
    ippAlphaIn,
    ippAlphaOut,
    ippAlphaATop,
    ippAlphaXor,
    ippAlphaPlus,
    ippAlphaOverPremul,
    ippAlphaInPremul,
    ippAlphaOutPremul,
    ippAlphaATopPremul,
    ippAlphaXorPremul,
    ippAlphaPlusPremul
} IppiAlphaType;
```

The `IppiDitherType` enumeration defines the type of dithering to be used by the [ippiReduceBits](#) function:

```
typedef enum {
    ippDitherNone,
    ippDitherFS,
    ippDitherJJN,
    ippDitherStucki,
    ippDitherBayer
} IppiDitherType;
```

The layout of the image slices used in some [image format conversion functions](#) is defined as follows:

```
enum {
    IPP_UPPER           = 1,
    IPP_LEFT            = 2,
    IPP_CENTER          = 4,
    IPP_RIGHT           = 8,
    IPP_LOWER           = 16,
    IPP_UPPER_LEFT      = 32,
    IPP_UPPER_RIGHT     = 64,
    IPP_LOWER_LEFT      = 128,
    IPP_LOWER_RIGHT     = 256
};
```

The `IppiShape` enumeration defines shapes of the structuring element used in some [morphological](#) functions:

```
typedef enum {
    ippiShapeRect      = 0,
    ippiShapeCross     = 1,
    ippiShapeEllipse    = 2,
    ippiShapeCustom     = 100
} IppiShape;
```

The `IppiAxis` enumeration defines the flip axes for the [ippiMirror](#) functions or direction of the image intensity ramp for the [ippiImageRamp](#) functions:

```
typedef enum {
    ippAxsHorizontal,
    ippAxsVertical,
    ippAxsBoth
} IppiAxis;
```

The `IppiBorderType` enumeration defines the border type that is used by some [“Separable Filters”](#) and [“Fixed Filters with Border”](#) functions:

```
typedef enum _IppiBorderType {
    ippBorderConst      = 0,
    ippBorderRepl       = 1,
    ippBorderWrap       = 2,
    ippBorderMirror     = 3,
    ippBorderMirrorR    = 4,
    ippBorderMirror2    = 5,
    ippBorderInMem      = 6,
    ippBorderInMemTop   = 0x0010,
    ippBorderInMemBottom = 0x0020
} IppiBorderType;
```

The `IppiWTSubband` enumeration defines the appropriate wavelet transform subband used in the [JPEG2000 coding functions](#):

```
typedef enum {
    ippWTSubbandLxLy,
    ippWTSubbandLxHy,
    ippWTSubbandHxLy,
    ippWTSubbandHxHy
} IppiWTSubband;
```

The `IppiMQTermination` enumeration defines how the MQ-coder will be terminated during the operation of the [JPEG2000 coding functions](#):

```
typedef enum {
    ippMQTermSimple,
    ippMQTermNearOptimal,
    ippMQTermPredictable
} IppiMQTermination;
```

The `IppiMQRateAppr` enumeration defines the padding-approximation model in the estimation of rate and distortion procedures performed by the [JPEG2000 coding functions](#):

```
typedef enum {
    ippMQRateApprGood
} IppiMQRateAppr;
```

The code flags used by the [JPEG2000 coding functions](#) are defined as follows:

```
enum
{
    IPP_JPEG2K_VERTICALLY_CAUSAL_CONTEXT,
    IPP_JPEG2K_SELECTIVE_MQ_BYPASS,
    IPP_JPEG2K_TERMINATE_ON_EVERY_PASS,
    IPP_JPEG2K_RESETCTX_ON_EVERY_PASS,
    IPP_JPEG2K_USE_SEGMENTATION_SYMBOLS,
    IPP_JPEG2K_LOSSLESS_MODE,
    IPP_JPEG2K_DEC_CONCEAL_ERRORS,
    IPP_JPEG2K_DEC_DO_NOT_CLEAR_CB,
    IPP_JPEG2K_DEC_DO_NOT_RESET_LOW_BITS,
    IPP_JPEG2K_DEC_DO_NOT_CLEAR_SFBUFFER,
    IPP_JPEG2K_DEC_CHECK_PRED_TERM
};
```

Some structures in the library are used to store function-specific (context) information. For example, the `IppiFFTSpec` structure stores twiddle factors and bit reverse indexes needed in computing the fast Fourier transform. Another examples are the structures used by the DFT and DCT functions.

These context-related structures are not defined in public headers, and the structure fields are not accessible to the user. It was done because the function context interpretation is processor dependent. Thus, you may only use context-related functions and may not create a function context as an automatic variable.

## Image Data Types and Ranges

The Intel IPP image processing functions support only absolute color images in which each pixel is represented by its channel intensities. The data storage for an image can be either pixel-oriented or plane-oriented (planar). For images in pixel order, all channel values for each pixel are clustered and stored consecutively, for example, RGBRGBRGB in case of an RGB image. The number of channels in a pixel-order image can be 1, 2, 3, or 4.

For images in planar order, all image data for each channel is stored contiguously followed by the next channel, for example, RRR...GGG...BBB.

Functions that operate on planar images are identified by the presence of `Pn` descriptor in their names. In this case, `n` pointers (one for each plane) may be specified.

The image data type is determined by the pixel depth in bits per channel, or bit depth. Bit depth for each channel can be 8, 16 or 32 and is included in the function name as one of these numbers (see [Function Naming](#) in this chapter for details). Some functions operate with images in 16-bit packed RGB format (see [RGB Image Formats](#) in Chapter 6 for more details). In this case data of all 3

channels are represented as 16u data type. The data may be signed (s), unsigned (u), or floating-point real (f). For some arithmetic and FFT/DFT functions, data in complex format (sc or fc) can be used, where each channel value is represented by two numbers: real and imaginary part. All channels in an image must have the same data type.

For example, in an absolute color 24-bit RGB image, three consecutive bytes (24 bits) per pixel represent the three channel intensities in pixel mode. This data type is identified in function names as 8u\_C3 descriptor, where 8u represents 8-bit unsigned data for each channel and C3 represents three channels.

For another example, in an absolute color 16-bit packed RGB image, two consecutive bytes (16 bits) per pixel represent the three channel intensities in pixel mode. This data type is identified in function names as 16u\_C3 descriptor, where 16u represents 16-bit unsigned data (not a bit depth) for all packed channels together and C3 stands for three channels.

If an alpha (opacity) channel is present in image data, the image must have four channels, with alpha channel being the last one. This data type is indicated by the AC4 descriptor. The presence of alpha channel can modify the function’s behavior. For such functions, Intel IPP provides versions with and without alpha. If an alpha channel is specified, the operation usually does not take place on that channel.

The range of values that can be represented by each data type lies between the lower and upper bounds. The following table lists data ranges and constant identifiers used in Intel IPP to denote the respective range bounds:

Table 2-2 Image Data Types and Ranges

Data Type	Lower Bound		Upper Bound	
	Identifier	Value	Identifier	Value
8s	IPP_MIN_8S	-128	IPP_MAX_8S	127
8u		0	IPP_MAX_8U	255
16s	IPP_MIN_16S	-32768	IPP_MAX_16S	32767
16u		0	IPP_MAX_16U	65535
32s	IPP_MIN_32S	-2 <sup>31</sup>	IPP_MAX_32S	2 <sup>31</sup> -1
32u		0	IPP_MAX_32U	2 <sup>32</sup> -1
32f <sup>†</sup>	IPP_MINABS_32F	1.175494351e <sup>-38</sup>	IPP_MAXABS_32F	3.402823466e <sup>38</sup>

<sup>†</sup> The range for absolute values

### Major Operation Models

Most Intel IPP image processing functions perform identical and independent operations on all channels of the processed image (except alpha channel). It means that the same operation is applied to each channel, and the computed results do not depend upon values of other channels. Some exceptions include the `ippiFilterMedianColor` function and color conversion functions, which process three channels together.

The Intel IPP image processing functions can be broken into two major models of operation: the functions that operate on one pixel to compute the result (also known as point operations), for example, `ippiAdd`, and the functions that operate on a group of pixels (also referred to as a neighborhood), for example, `ippiFilterBox`.

### Neighborhood operations

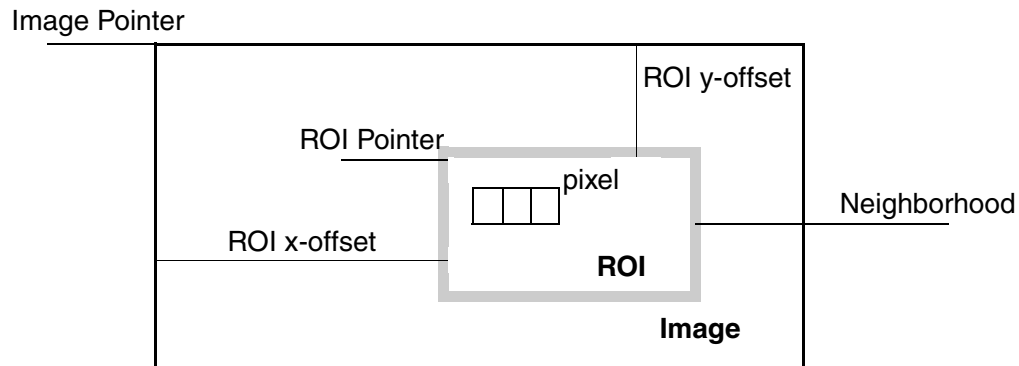
The result of a neighborhood operation is based on values of a certain group of pixels, located near a given input pixel. The set of neighboring pixels is typically defined by a rectangular mask (or kernel) and anchor cell, specifying the mask alignment with respect to the position of the input pixel.

The Intel IPP functions that process a neighborhood operate on the assumption that all referred points of the image are available. To support this mode, the application must check that ROI parameters passed to the function have such values that all processed neighborhood pixels actually exist in the image.

### Regions of Interest in Intel IPP

Most Intel IPP image processing functions can operate not only on entire images but also on image areas. Image region of interest (ROI) is a rectangular area that may be either some part of the image or the whole image.

Intel IPP functions with ROI support are distinguished by the presence of an `R` descriptor in their names. ROI of an image is defined by the size and offset from the image origin as shown in [Figure 2-1](#). The origin of an image is implied to be in the top left corner, with *x* values increasing from left to right and *y* values increasing downwards.

**Figure 2-1 Image, ROI, and Offsets**

Both the source and destination images can have a region of interest. In such cases the sizes of ROIs are assumed to be the same while offsets may differ. The image processing is then performed on data of the source ROI, and the results are written to the destination ROI.

In function call sequences, an ROI is specified by:

- *roiSize* argument of the *IppiSize* type
- *pSrc* and *pDst* pointers to the starts of source and destination ROI buffers
- *srcStep* and *dstStep* arguments which are equal to distances in bytes between the starts of consecutive lines in source and destination images, respectively.

Thus, the arguments *srcStep*, *dstStep* set steps in bytes through image buffers to start processing a new line in the ROI of an image.



The following code example illustrates the use of the *dstStep* parameter in function calls:

## Example 2-1 Using Buffer Step Parameter

---

```

IppStatus alignedLine( void ) {
    Ipp8u x[8*3] = {0};
    IppiSize imgSize = {5,3};
    /// The image is of size 5x3. Width 8 has been
    /// chosen by the user to align every line of the image
    return ippiSet_8u_C1R( 7, x, 8, imgSize);
}

```

The resultant image *x* contains the following data:

```

07 07 07 07 07 00 00 00
07 07 07 07 07 00 00 00
07 07 07 07 07 00 00 00

```

---

If ROI is present,

- source and destination images can have different sizes
- lines may have padding at the end for aligning the line sizes
- application must correctly define the *pSrc*, *pDst* and *roiSize* arguments

The *pSrc* and *pDst* arguments are the shifted pointers to the image data. For example, in case of ROI operations on 3-bytes-per-pixel image data (8u\_C3R), *pSrc* points to the start of the source ROI buffer and can be interpreted as follows:

*pSrc* = *pSrcImg* + 3 \* (*srcImgSize.width* \* *srcRoiOffset.y* + *srcRoiOffset.x*),

where

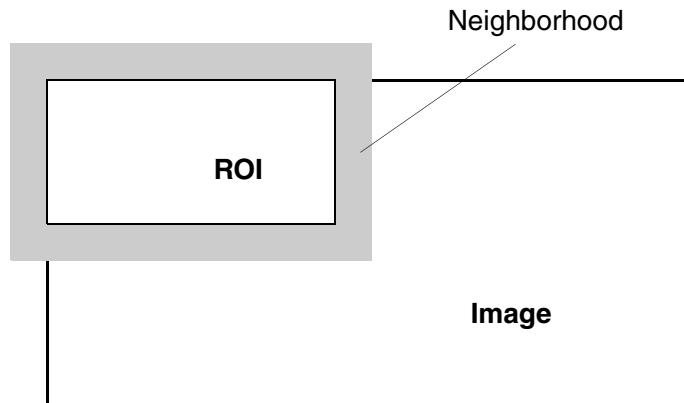
*pSrcImg* points to the start of the source image buffer;

*srcImgSize* is the image size in pixels (of the *IppiSize* type);

*srcRoiOffset* determines an offset of ROI relative to the start of the image as shown in [Figure 2-1](#).

For functions using ROI with a neighborhood, you should correctly use values of the *pSrc* and *roiSize* parameters. These functions assume that the points in the neighborhood exist and that therefore *pSrc* is almost never equal to *pSrcImg*. [Figure 2-2](#) illustrates the case when neighborhood pixels can fall outside the source image.

**Figure 2-2** Using ROI with Neighborhood



To ensure valid operation when image pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).



**WARNING.** If the required border pixels are not defined prior to calling neighborhood functions that attempt to process such pixels, you may get memory violation errors.

The following code example shows how to process an image with ROI:

### **Example 2-2 ROI Processing**

---

```
IppStatus roi( void ) {  
    Ipp8u x[8*3] = {0};  
    IppiSize roiSize = {3,2};  
    IppiPoint roiPoint = {2,1};  
    /// place the pointer to the ROI start position  
    return ippiset_8u_C1R( 7, x+8*roiPoint.y+roiPoint.x, 8,  
        roiSize );  
}
```

The destination image *x* will contain the following data

```
00 00 00 00 00 00 00 00  
00 00 07 07 07 00 00 00  
00 00 07 07 07 00 00 00
```

---

### **Tiled Image Processing**

The Intel IPP can process images composed from tiles, or tiled images. The specially created code sample “Tiled Image processing” demonstrates how to do it. See *Intel IPP Image Processing Samples* downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>.

# Support Functions

## 3

This chapter describes Intel IPP support functions that are used to:

- retrieve information about the current Intel IPP software version
- get a brief explanation of the returned status codes
- allocate and free memory that is needed for the operation of other Intel IPP image and video processing functions.

The list of support functions is given in [Table 3-1](#).

**Table 3-1 Intel IPP Support Functions**

Function Base Name	Operation
<b>Version Information Functions</b>	
<a href="#">GetLibVersion</a>	Returns information about the active library version.
<b>Status Information Function</b>	
<a href="#">ippGetStatusString</a>	Translates a status code into a message.
<b>Memory Allocation Functions</b>	
<a href="#">Malloc</a>	Allocates memory aligned to 32-byte boundary.
<a href="#">Free</a>	Frees memory allocated by the function <code>ippiMalloc</code> .

## Version Information Function

This function returns the version number and other information about the active Intel IPP image processing software.

---

### GetLibVersion

*Returns information about the used version of Intel IPP software for image processing.*

---

#### Syntax

```
const IppLibraryVersion* ippiGetLibVersion(void);
```

#### Description

The function `ippiGetLibVersion` is declared in the `ippi.h` file. This function returns a pointer to a static data structure `IppLibraryVersion` that contains information about the current version of the Intel IPP for image processing. There is no need for you to release memory referenced by the returned pointer, as it points to a static variable. The following fields of the `IppLibraryVersion` structure are available:

<i>major</i>	the major number of the current library version.
<i>minor</i>	the minor number of the current library version.
<i>Name</i>	the name of the current library version.
<i>Version</i>	the library version string.
<i>BuildDate</i>	the library version actual build date.

For example, if the library version is “v1.2 Beta”, library name is “ippim6”, and build date is “Jul 20 99”, then the fields in this structure are set as:

```
major=1, minor=2, Name=“ippim6”,
Version=“v1.2 Beta”, BuildDate=“Jul 20 99”
```

[Example 3-1](#) shows how to use the function `ippiGetLibVersion`.

**Example 3-1 Using theippiGetLibVersion Function**

---

```
void libinfo(void) {  
    const IppLibraryVersion* lib =ippiGetLibVersion();  
    printf("%s %s %d.%d.%d.%d\n", lib->Name, lib->Version,  
        lib->major, lib->minor, lib->majorBuild, lib->build);  
}
```

Output:

```
ippia6 v0.0 Alpha 0.0.5.5
```

---



---

**NOTE.** Each sub-library that is used in the image processing domain has its own similar function to retrieve information about the active library version. These functions are: `ippGetLibVersion`, `ippjGetLibVersion`, `ippcvGetLibVersion`, `ippvcGetLibVersion`, and `ippccGetLibVersion`. They are declared in the following header files: `ippcore.h`, `ippj.h`, `ippcv.h`, `ippvc.h`, `ippcc.h`, respectively, and have the same interface as the above described function.

---

## Status Information Function

Use this function to get a brief description of the status code returned by the current Intel IPP software.

---

### ippGetStatusString

*Translates a status code into a message.*

---

#### Syntax

```
const char* ippGetStatusString(IppStatus StsCode);
```

#### Parameters

*StsCode*                      Code that indicates the status type (see [Table 2-1](#))

#### Description

The function `ippGetStatusString` is declared in the `ipps.h` file. This function returns a pointer to the text string associated with a status code *StsCode*. Use this function to produce error and warning messages for users. The returned pointer is a pointer to an internal static buffer and need not be released.

A code example below shows how to use the function `ippGetStatusString`. If you call an Intel IPP function, in this example `ippiSet_8u_C1R`, with a NULL pointer, it returns an error code -8.

The status information function translates this code into the corresponding message “Null Pointer Error”:

Example 3-2 Using the Status Information Function

```
void statusInfo( void ) {
    IppiSize roi = {0};
    IppStatus st = ippiSet_8u_C1R( 3, 0, 0, roi );
    printf( " %d : %s\n", st, ippiGetStatusString( st ) );
}
```

Output:  
-8, Null Pointer Error

Memory Allocation Functions

This section describes the Intel IPP functions that allocate aligned memory blocks for data of required type, or free the previously allocated memory.



**NOTE.** The only function to free the memory allocated by any of these functions is `ippiFree()`.

Malloc

*Allocates memory aligned to 32-byte boundary.*

Syntax

```
Ipp<datatype>* ippiMalloc_<mod>(int widthPixels, int heightPixels,
    int* pStepBytes);
```

Supported values for *mod* :

8u_C1	16u_C1	16s_C1	32s_C1	32f_C1	32sc_C1	32fc_C1
8u_C2	16u_C2	16s_C2	32s_C2	32f_C2	32sc_C2	32fc_C2
8u_C3	16u_C3	16s_C3	32s_C3	32f_C3	32sc_C3	32fc_C3



8u_C4	16u_C4	16s_C4	32s_C4	32f_C4	32sc_C4	32fc_C4
8u_AC4	16u_AC4	16s_AC4	32s_AC4	32f_AC4	32sc_AC4	32fc_AC4

## Parameters

<i>widthPixels</i>	Width of an image in pixels.
<i>heightPixels</i>	Height of an image in pixels.
<i>pStepBytes</i>	Pointer to the step in bytes through the image.

## Description

The function `ippiMalloc` is declared in the `ippi.h` file. This function allocates a memory block aligned to a 32-byte boundary for elements of different data types. Every line of the image is aligned by padding with zeros in accordance with the *pStepBytes* parameter, which is calculated by the function `ippiMalloc` and returned for further use.

Note that the function `ippiMalloc` allocates one continuous memory block, whereas functions that operate on planar images require an array of separate pointers (`IppType* plane[3]`) to each plane as input. The following code example demonstrates how to construct such an array and set correct values to the pointers in order to use the allocated memory block with Intel IPP functions operating on planar images. The example is given for `16s` data type.

### Example 3-3 Setting Values to the Pointers to Image Planes

---

```
int stepBytes;
Ipp16s* plane[3];

Ipp16s* img = ippiMalloc_16s_P3( widthPixels, heightPixels,
&stepBytes );
plane[0] = img;
plane[1] = (Ipp16s*)((Ipp8u*)img + heightPixels * stepBytes);
plane[2] = (Ipp16s*)((Ipp8u*)img + 2 * heightPixels * stepBytes);
```

---

## Return Values

The return value of `ippiMalloc` function is a pointer to an aligned memory block. If no memory is available in the system, the `NULL` value is returned. To free the allocated memory block, the function `ippiFree` must be used.

---

## Free

*Frees memory allocated by the function `ippiMalloc`.*

---

### Syntax

```
void ippiFree(void* ptr);
```

### Parameters

*ptr*            Pointer to a memory block to be freed. This block must have been previously allocated by the function `ippiMalloc`.

### Description

The function `ippiFree` is declared in the `ippi.h` file. This function frees the aligned memory block allocated by the function `ippiMalloc`.



---

**NOTE.** The function `ippiFree` cannot be used to free memory allocated by standard functions like `malloc` or `calloc`, nor can the memory allocated by `ippiMalloc` be freed by `free`.

---

# Image Data Exchange and Initialization Functions

## 4

This chapter describes Intel IPP image processing functions that perform image data manipulation, exchange and initialization operations.

[Table 4-1](#) lists functions described in more detail later in this chapter:

**Table 4-1 Image Data Exchange and Initialization Functions**

Function Base Name	Operation
<a href="#">Convert</a>	Converts pixel values in an image from one bit depth to another
<a href="#">Scale</a>	Scales pixel values of a image and converts them to another bit depth
<a href="#">Scale</a>	Sets an array of pixels to a value
<a href="#">Copy</a>	Copies pixel values between two images
<a href="#">CopyConstBorder</a>	Copies pixels values between two images and adds the border pixels with a constant value
<a href="#">CopyReplicateBorder</a>	Copies pixel values between two images and adds the replicated border pixels
<a href="#">CopyWrapBorder</a>	Copies pixel values between two images and adds the border pixels
<a href="#">CopySubpix</a>	Copies pixel values between two images with subpixel precision.
<a href="#">CopySubpixIntersect</a>	Copies pixel values of the intersection with specified window with subpixel precision.
<a href="#">Dup</a>	Copies gray scale image to all channels of the color image.
<a href="#">Transpose</a>	Transpose a source image.
<a href="#">SwapChannels</a>	Changes the order of channels of the image
<a href="#">AddRandUniform Direct</a>	Generates random samples with uniform distribution and adds them to an image data

**Table 4-1 Image Data Exchange and Initialization Functions (continued)**

Function Base Name	Operation
<a href="#">AddRandGauss_Direct</a>	Generates random samples with Gaussian distribution and adds them to an image data
<a href="#">ImageJaehne</a>	Creates the Jaehne's test image
<a href="#">ImageRamp</a>	Creates the test image that has an intensity ramp
<a href="#">SampleLine</a>	Reads all pixels on the raster line into the buffer
<a href="#">ZigzagFwd8x8</a>	Converts a conventional order to the zigzag sequence.
<a href="#">ZigzagInv8x8</a>	Converts a zigzag sequence to the conventional order.

Many solutions and hints for use of these functions can be found in Intel IPP Samples. See *Intel IPP Image Processing and Media Processing Samples* downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>.

## Convert

*Converts pixel values of an image from one bit depth to another.*

### Syntax

#### Case 1: Conversion to increased bit depth

```
ippiStatus ippiConvert_1u8u_C1R(const Ipp8u* pSrc, int srcStep,
    int srcBitOffset, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
ippiStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u16u_C1R	8u16s_C1R	8u32s_C1R	8s32s_C1R	16u32s_C1R
8u16u_C3R	8u16s_C3R	8u32s_C3R	8s32s_C3R	16u32s_C3R
8u16u_C4R	8u16s_C4R	8u32s_C4R	8s32s_C4R	16u32s_C4R
8u16u_AC4R	8u16s_AC4R	8u32s_AC4R	8s32s_AC4R	16u32s_AC4R
8u32f_C1R	8s32f_C1R	16u32f_C1R	16s32f_C1R	

8u32f_C3R	8s32f_C3R	16u32f_C3R	16s32f_C3R
8u32f_C4R	8s32f_C4R	16u32f_C4R	16s32f_C4R
8u32f_AC4R	8s32f_AC4R	16u32f_AC4R	16s32f_AC4R

## Case 2: Conversion to reduced bit depth: integer to integer type

```
IppStatus ippiConvert_8u1u_C1R( const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize,
    Ipp8u threshold);
```

```
IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

16u8u_C1R	16s8u_C1R	32s8u_C1R	32s8s_C1R
16u8u_C3R	16s8u_C3R	32s8u_C3R	32s8s_C3R
16u8u_C4R	16s8u_C4R	32s8u_C4R	32s8s_C4R
16u8u_AC4R	16s8u_AC4R	32s8u_AC4R	32s8s_AC4R

## floating point to integer type

```
IppStatus ippiConvert_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,
    IppRoundMode roundMode);
```

Supported values for *mod*:

32f8u_C1R	32f8s_C1R	32f16u_C1R	32f16s_C1R
32f8u_C3R	32f8s_C3R	32f16u_C3R	32f16s_C3R
32f8u_C4R	32f8s_C4R	32f16u_C4R	32f16s_C4R
32f8u_AC4R	32f8s_AC4R	32f16u_AC4R	32f16s_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcBitOffset</i>	Offset (in bits) in the first byte of the source image row.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstBitOffset</i>	Offset (in bits) in the first byte of the destination image row.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	Threshold level for Stucki's dithering.
<i>roundMode</i>	Rounding mode which can be <code>ippRndZero</code> or <code>ippRndNear</code> :
<code>ippRndZero</code>	Specifies that floating-point values must be truncated toward zero.
<code>ippRndNear</code>	Specifies that floating-point values must be rounded to the nearest integer.

## Description

The function `ippiConvert` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts pixel values in the source image ROI *pSrc* to a different bit depth and writes them to the destination image ROI *pDst*. No scaling is done. When converting from floating-point to integer type, rounding defined by *roundMode* is performed, and the result is saturated to the destination data type range.

**1u to 8u conversion.** The source image has a `8u` data type, where each byte represents eight consecutive pixels of the bitonal image (1 bit per pixel). In this case, additional parameter *srcBitOffset* is required to specify the start position of the source ROI buffer.

Each source bit is transformed into the byte of the destination image in the following way: if an input pixel is 0, the corresponding output pixel is set to 0, if an input pixel is 1, the corresponding output pixel is set to 255. Note that in the source image the bit order in each byte is inverse relative to the pixel order, that is, the first pixel in a row is represented by the last (7th) bit of the first byte in the row.

**8u to 1u conversion.** Source image is converted to a bitonal image using the Stucki's error diffusion dithering algorithm.

The destination image has a `8u` data type, where each byte represents eight consecutive pixels of the bitonal image (1 bit per pixel). In this case, additional parameter *dstBitOffset* is required to specify the start position of the destination ROI buffer.

[Example 4-1](#) illustrates data conversion without scaling.

**Example 4-1    Straightforward Data Conversion**

```

IppStatus convert( void ) {
    IppiSize roi = {5,4};
    Ipp32f x[5*4];
    Ipp8u y[5*4];
    ippiSet_32f_C1R( -1.0f, x, 5*sizeof(Ipp32f), roi );
    x[1] = 300; x[2] = 150;
    return ippiConvert_32f8u_C1R( x, 5*sizeof(Ipp32f), y, 5, roi, ippRndNear );
}

```

The destination image *y* contains:

```

00 FF 96 00 00
00 00 00 00 00
00 00 00 00 00
00 00 00 00 00

```

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or <i>srcBitOffset/dstBitOffset</i> is less than zero.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## Scale

*Scales pixel values of an image and converts them to another bit depth.*

---

### Syntax

#### Case 1: Scaling with conversion to integer data of increased bit depth

```
IppStatus ippiScale_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u16u_C1R	8u16s_C1R	8u32s_C1R
8u16u_C3R	8u16s_C3R	8u32s_C3R
8u16u_C4R	8u16s_C4R	8u32s_C4R
8u16u_AC4R	8u16s_AC4R	8u32s_AC4R

#### Case 2: Scaling with conversion to floating-point data

```
IppStatus ippiScale_<mod>(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for *mod*:

8u32f\_C1R  
8u32f\_C3R  
8u32f\_C4R  
8u32f\_AC4R

#### Case 3: Scaling of integer data with conversion to reduced bit depth

```
IppStatus ippiScale_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,
    IppHintAlgorithm hint);
```

Supported values for *mod*:

16u8u_C1R	16s8u_C1R	32s8u_C1R
16u8u_C3R	16s8u_C3R	32s8u_C3R
16u8u_C4R	16s8u_C4R	32s8u_C4R
16u8u_AC4R	16s8u_AC4R	32s8u_AC4R



**Case 4: Scaling of floating-point data with conversion to integer data type**

```
IppStatus ippiScale_<mod>(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for *mod*:

```
32f8u_C1R
32f8u_C3R
32f8u_C4R
32f8u_AC4R
```

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>vMin</i> , <i>vMax</i>	Minimum and maximum values of the input data.
<i>hint</i>	Option to select the algorithmic implementation of the function.

**Description**

The function `ippiScale` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts pixel values of a source image ROI *pSrc* to the destination data type, using a linear mapping. Computation algorithm is specified by the *hint* argument (see [Table 11-3](#)). For conversions between integer data types, the whole range [*src\_Min*..*src\_Max*] of the input data type is mapped onto the range [*dst\_Min*..*dst\_Max*] of the output data type (see chapter 2 for more information on data types and ranges).

The following scaling formula to map the source pixel *p* to the destination pixel *p'* is used:

$$p' = dst\_Min + k * (p - src\_Min)$$

where  $k = (dst\_Max - dst\_Min) / (src\_Max - src\_Min)$

For conversions to and from floating-point data type, the user-defined floating-point data range [*vMin*..*vMax*] is mapped onto the source or destination data type range.

If the conversion is from `Ipp32f` type and some of the input floating-point values are outside the specified input data range `[vMin..vMax]`, the corresponding output values will saturate. To determine the actual floating-point data range in your image, use the function `ippiMinMax`.

[Example 4-2](#) shows how to use scaling to preserve the data range:

## Example 4-2 Data Conversion with Scaling

---

```
IppStatus scale( void ) {
    IppiSize roi = {5,4};
    Ipp32f x[5*4];
    Ipp8u y[5*4];
    ippiSet_32f_C1R( -1.0f, x, 5*sizeof(Ipp32f), roi );
    x[1] = 300; x[2] = 150;
    return ippiScale_32f8u_C1R( x, 5*sizeof(Ipp32f), y, 5, roi, -1, 300 );
}
```

The destination image `y` contains:

```
00 FF 80 00 00
00 00 00 00 00
00 00 00 00 00
00 00 00 00 00
```

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsScaleRangeErr</code>	Indicates an error condition if the input data bounds are incorrect, that is <code>vMax</code> is less than or equal to <code>vMin</code> .

## Set

*Sets an array of pixels to a value.*

### Syntax

#### Case 1: Setting one-channel data to a value

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16s_C1R	32s_C1R	32f_C1R
--------	---------	---------	---------

#### Case 2: Setting each color channel to a specified value

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[3],
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

#### Case 3: Setting color channels and alpha channel to specified values

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[4],
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C4R	16s_C4R	32s_C4R	32f_C4R
--------	---------	---------	---------

#### Case 4: Setting masked one-channel data to a value

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for *mod*:

8u_C1MR	16s_C1MR	32s_C1MR	32f_C1MR
---------	----------	----------	----------

#### Case 5: Setting color channels of masked multi-channel data to specified values

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for *mod*:

8u_C3MR	16s_C3MR	32s_C3MR	32f_C3MR
8u_AC4MR	16s_AC4MR	32s_AC4MR	32f_AC4MR

### Case 6: Setting all channels of masked multi-channel data to specified values

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for *mod*:

8u_C4MR	16s_C4MR	32s_C4MR	32f_C4MR
---------	----------	----------	----------

### Case 7: Setting selected channel of multi-channel data to a value

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3CR	16s_C3CR	32s_C3CR	32f_C3CR
8u_C4CR	16s_C4CR	32s_C4CR	32f_C4CR

## Parameters

<i>value</i>	Constant value assigned to each pixel in the destination image ROI.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>pMask</i>	Pointer to the mask image buffer.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image buffer.

## Description

The function `ippiSet` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets pixels in the destination image ROI *pDst* to a constant *value*. Either all pixels in a rectangular ROI, or only those selected by the specified mask *pMask*, can be set to a value. In case of masked operation, the function sets pixel values in the destination buffer only if the spatially corresponding mask array value is non-zero.

When a channel of interest is selected, that is only one channel of a multi-channel image must be set (see **Case 7**), the *pDst* pointer points to the start of ROI buffer in the required channel. If alpha channel is present in the source image data, the alpha components may be either skipped, or set to a value, depending on the chosen *ippiSet* function flavor.

### Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippiStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippiStsStepErr</code>	Indicates an error condition if <i>dstStep</i> or <i>maskStep</i> has a zero or negative value.

---

## Copy

*Copies pixel values between two images.*

---

### Syntax

#### Case 1: Copying all pixels of all color channels

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16s_C1R</code>	<code>32s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>	<code>32s_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>16s_C4R</code>	<code>32s_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>	<code>32s_AC4R</code>	<code>32f_AC4R</code>
<code>8u_C3AC4R</code>	<code>16s_C3AC4R</code>	<code>32s_C3AC4R</code>	<code>32f_C3AC4R</code>
<code>8u_AC4C3R</code>	<code>16s_AC4C3R</code>	<code>32s_AC4C3R</code>	<code>32f_AC4C3R</code>

#### Case 2: Operation on masked pixels only

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,  
    const Ipp8u* pMask, int maskStep);
```

Supported values for *mod*:

8u_C1MR	16s_C1MR	32s_C1MR	32f_C1MR
8u_C3MR	16s_C3MR	32s_C3MR	32f_C3MR
8u_C4MR	16s_C4MR	32s_C4MR	32f_C4MR
8u_AC4MR	16s_AC4MR	32s_AC4MR	32f_AC4MR

### Case 3: Copying of a selected channel in a multi-channel image

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3CR	16s_C3CR	32s_C3CR	32f_C3CR
8u_C4CR	16s_C4CR	32s_C4CR	32f_C4CR

### Case 4: Copying of a selected channel to a one-channel image

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3C1R	16s_C3C1R	32s_C3C1R	32f_C3C1R
8u_C4C1R	16s_C4C1R	32s_C4C1R	32f_C4C1R

### Case 5: Copying a one-channel image to a multi-channel image

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1C3R	16s_C1C3R	32s_C1C3R	32f_C1C3R
8u_C1C4R	16s_C1C4R	32s_C1C4R	32f_C1C4R

### Case 6: Splitting color image into separate planes

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* const pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3P3R	16s_C3P3R	32s_C3P3R	32f_C3P3R
----------	-----------	-----------	-----------

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C4P4R      16s\_C4P4R      32s\_C4P4R      32f\_C4P4R

### Case 7: Composing color image from separate planes

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* const pSrc[3], int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_P3C3R      16s\_P3C3R      32s\_P3C3R      32f\_P3C3R

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_P4C4R      16s\_P4C4R      32s\_P4C4R      32f\_P4C4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI. The array storing pointers to the ROIs in the separate color planes of the source image for <b>Case 7</b> .
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI. The array storing pointers to the ROIs in the resulting color planes for <b>Case 6</b> .
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>pMask</i>	Pointer to the mask image buffer.
<i>maskStep</i>	Sep in bytes through the mask image buffer.

### Description

The function `ippiCopy` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies data from a source image ROI *pSrc* to the destination image ROI *pDst*. Copying of pixels selected by a mask *pMask* is supported as well. In case of masked operation, the function writes pixel values in the destination buffer only if the spatially corresponding mask array value is non-zero (as illustrated in the code example below).

Function flavors with channel of interest descriptor (C) copy only one specified channel of a source multi-channel image to another multi-channel image (see **Case 3**). For these functions, the *pSrc* and *pDst* pointers point to the starts of ROI buffers in the specified channels of source and destination images, respectively.

Function flavors exist that can be used to copy data from only one specified channel *pSrc* of a multi-channel image to a one-channel image *pDst* (see **Case 4**), as well as to copy data from a one-channel image *pSrc* to only one specified channel of a multi-channel image *pDst* (see **Case 5**).

The function `ippiCopy` can also be used to split the color image into separate planes (see **Case 6**).

[Example 4-3](#) below shows how to copy masked data:

### Example 4-3 Copying Masked Data

---

```
IppStatus copyWithMask( void ) {
    Ipp8u mask[3*3], x[5*4], y[5*4]={0};
    IppiSize imgroi={5,4}, mskroi={3,3};
    ippiSet_8u_C1R( 3, x, 5, imgroi );
    /// set mask with a hole in upper left corner
    ippiSet_8u_C1R( 1, mask, 3, mskroi );
    mask[0] = 0;
    /// copy roi with mask
    return ippiCopy_8u_C1MR( x, 5, y, 5, mskroi, mask, 3 );
}
```

The destination image *y* contains:

```
00 03 03 00 00
03 03 03 00 00
03 03 03 00 00
00 00 00 00 00
```

---

The function `ippiCopy_8u_C1R` is used in H.264 decoder and encoder included into Intel® IPP Samples downloadable from

<http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>.



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> , with the exception of second mode in <b>Case 4</b> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> for <b>Cases 4</b> and <b>5</b> , if .

---

## CopyConstBorder

*Copies pixels values between two images and adds the border pixels with a constant value.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippCopyConstBorder_<mod>(const Ipp<datatype>* pSrc,  
    int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep,  
    IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth,  
    Ipp<datatype> value);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16s_C1R</code>	<code>32s_C1R</code>
---------------------	----------------------	----------------------

#### Case 2: Operation on multi-channel data

```
IppStatus ippCopyConstBorder_<mod>(const Ipp<datatype>* pSrc,  
    int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep,  
    IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth, const  
    Ipp<datatype> value[3]);
```

Supported values for `mod`:

<code>8u_C3R</code>	<code>16s_C3R</code>	<code>32s_C3R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>	<code>32s_AC4R</code>

```

IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep,
    IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth,
    const Ipp<datatype> value[4]);

```

Supported values for *mod*:

8u\_C4R      16s\_C4R      32s\_C4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>topBorderHeight</i>	Height of the top border in pixels.
<i>leftBorderWidth</i>	Width of the left border in pixels.
<i>value</i>	The constant value to assign to the border pixels (constant vector in case of multi-channel images).

## Description

The function `ippiCopyConstBorder` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies the source image *pSrc* to the destination image *pDst* and creates border outside the copied area; pixel values of the border are set to the specified constant value that is passed by the *value* argument. The parameters *topBorderHeight* and *leftBorderWidth* specify the position of the first pixel of the source ROI in the destination image ROI (see [Figure 4-1](#).) Squares marked in red correspond to pixels copied from the source image, that is, the source image ROI.

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the *topBorderHeight* (*leftBorderWidth*) parameter.

**Figure 4-1**      **Creating a Border of Pixels with Constant Value**

v	v	v	v	v	v	v	v	v
v	v	v	v	v	v	v	v	v
v	v	1	2	3	4	5	v	v
v	v	6	7	8	9	10	v	v
v	v	11	12	13	14	15	v	v
v	v	16	17	18	19	20	v	v
v	v	v	v	v	v	v	v	v

*topBorderHeight=2*  
*leftBorderWidth=2*

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with zero or negative value, or <i>topBorderHeight</i> or <i>leftBorderWidth</i> is less than zero, or <i>dstRoiSize.width</i> < <i>srcRoiSize.width</i> + <i>leftBorderWidth</i> , or <i>dstRoiSize.height</i> < <i>srcRoiSize.height</i> + <i>topBorderHeight</i> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## CopyReplicateBorder

*Copies pixel values between two images and adds the replicated border pixels.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep,
    IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32s_C1R
8u_C3R	16s_C3R	32s_C3R
8u_C4R	16s_C4R	32s_C4R
8u_AC4R	16s_AC4R	32s_AC4R

#### Case 2: In-place operation

```
IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc,
    int srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int
    topBorderHeight, int leftBorderWidth);
```

Supported values for *mod* :

8u_C1IR	16s_C1IR	32s_C1IR
8u_C3IR	16s_C3IR	32s_C3IR
8u_C4IR	16s_C4IR	32s_C4IR
8u_AC4IR	16s_AC4IR	32s_AC4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.

<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>topBorderHeight</i>	Height of the top border in pixels.
<i>leftBorderWidth</i>	Width of the left border in pixels.

## Description

The function `ippiCopyReplicateBorder` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies the source image ROI *pSrc* to the destination image ROI *pDst* and fills pixels (“border”) outside the copied area in the destination image with the values of the source image pixels according to the scheme illustrated in the [Figure 4-2](#). Squares marked in red correspond to pixels copied from the source image.

Note that the in-place function flavor actually adds border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters *topBorderHeight* and *leftBorderWidth* specify the position of the first pixel of the source ROI in the destination image ROI.

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the *topBorderHeight* (*leftBorderWidth*) parameter. .

**Figure 4-2**      **Creating a Replicated Border**

1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
1	1	1	2	3	4	5	5	5
6	6	6	7	8	9	10	10	10
11	11	11	12	13	14	15	15	15
16	16	16	17	18	19	20	20	20
16	16	16	17	18	19	20	20	20

*topBorderHeight=2*  
*leftBorderWidth=2*

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with zero or negative value, or <i>topBorderHeight</i> or <i>leftBorderWidth</i> is less than zero, or <i>dstRoiSize.width</i> < <i>srcRoiSize.width</i> + <i>leftBorderWidth</i> , or <i>dstRoiSize.height</i> < <i>srcRoiSize.height</i> + <i>topBorderHeight</i> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

## CopyWrapBorder

*Copies pixel values between two images and adds the border pixels.*

---

### Syntax

```
IppStatus ippCopyWrapBorder_32s_C1R(const Ipp32s* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp32s* pDst, int dstStep,
    IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
IppStatus ippCopyWrapBorder_32s_C1R(const Ipp32s* pSrc,
    int srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize,
    int topBorderHeight, int leftBorderWidth);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for in-place flavor.
<i>topBorderHeight</i>	Height of the top border in pixels.
<i>leftBorderWidth</i>	Width of the left border in pixels.

## Description

The function `ippiCopyWrapBorder` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies the source image ROI *pSrc* to the destination image *pDst* and fills pixels (“border”) outside the copied area in the destination image with the values of the source image pixels according to the scheme illustrated in the [Figure 4-3](#). Squares marked in red correspond to pixels copied from the source image.

Note that the in-place function flavor actually adds border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters *topBorderHeight* and *leftBorderWidth* specify the position of the first pixel of the source ROI in the destination image ROI.

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the *topBorderHeight* (*leftBorderWidth*) parameter.

**Figure 4-3** Creating a Border of Pixels by `ippiCopyWrapBorder` Function

14	15	11	12	13	14	15	11	12	13	14	15	11
19	20	16	17	18	19	20	16	17	18	19	20	16
4	5	1	2	3	4	5	1	2	3	4	5	1
9	10	6	7	8	9	10	6	7	8	9	10	6
14	15	11	12	13	14	15	11	12	13	14	15	11
19	20	16	17	18	19	20	16	17	18	19	20	16
4	5	1	2	3	4	5	1	2	3	4	5	1
9	10	6	7	8	9	10	6	7	8	9	10	6
14	15	11	12	13	14	15	11	12	13	14	15	11
19	20	16	17	18	19	20	16	17	18	19	20	16
4	5	1	2	3	4	5	1	2	3	4	5	1
9	10	6	7	8	9	10	6	7	8	9	10	6
14	15	11	12	13	14	15	11	12	13	14	15	11
19	20	16	17	18	19	20	16	17	18	19	20	16
4	5	1	2	3	4	5	1	2	3	4	5	1
9	10	6	7	8	9	10	6	7	8	9	10	6

*topBorderHeight=2*  
*leftBorderWidth=2*

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with zero or negative value, or <i>topBorderHeight</i> or <i>leftBorderWidth</i> is less than zero, or <i>dstRoiSize.width</i> < <i>srcRoiSize.width</i> + <i>leftBorderWidth</i> , or <i>dstRoiSize.height</i> < <i>srcRoiSize.height</i> + <i>topBorderHeight</i> .
<code>ippStsStepErr</code>	Indicates an error condition if one of <i>srcStep</i> , <i>dstStep</i> , <i>srcDstStep</i> has a zero or negative value.



## CopySubpix

*Copies pixel values between two images with subpixel precision.*

### Syntax

#### Case 1: Copying without conversion or with conversion to floating point data

```
IppStatus ippiCopySubpix_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp32f dx, Ipp32f dy);
```

Supported values for *mod*:

```
8u_C1R      16u_C1R      32f_C1R
8u32f_C1R   16u32f_C1R
```

#### Case 2: Copying with conversion to integer data

```
IppStatus ippiCopySubpix_8u16u_C1R_Sfs(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy,
    int scaleFactor);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>dx</i>	Fractional part of the x-coordinate in the source image.
<i>dy</i>	Fractional part of the y-coordinate in the source image.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiCopySubpix` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the pixel value of the destination image using linear interpolation (see [“Linear Interpolation” on page B-3](#)) in accordance with the following formulas:

$pDst_{j,i} = pSrc_{j+dx,i+dy}$ , where  $i=0, \dots, roiSize.height-1$ ,  $j=0, \dots, roiSize.width-1$ .

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width*<i>&lt;pixelSize&gt;</i></code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of <code>srcStep</code> or <code>dstStep</code> is not divisible by 4 for floating point images, or by 2 for short integer images.

---

## CopySubpixIntersect

*Copies pixel values of the intersection with specified window with subpixel precision.*

---

### Syntax

#### Case 1: Copying without conversion or with conversion to floating point data

```
IppStatus ippiCopySubpixIntersect_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, Ipp<dstDatatype>* pDst, int
    dstStep, IppiSize dstRoiSize, IppiPoint_32f point, IppiPoint* pMin,
    IppiPoint* pMax);
```

Supported values for `mod`:

`8u_C1R`      `16u_C1R`      `32f_C1R`

8u32f\_C1R    16u32f\_C1R

## Case 2: Copying with conversion to integer data

```
IppStatus ippiCopySubpixIntersect_8u16u_C1R_Sfs(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp16u* pDst, int dstStep, IppiSize
    dstRoiSize, IppiPoint_32f point, IppiPoint* pMin, IppiPoint* pMax,
    int scaleFactor);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source image ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination image ROI in pixels.
<i>point</i>	Center point of the window.
<i>pMin</i>	Pointer to coordinates of the top left pixel of the intersection in the destination image.
<i>pMax</i>	Pointer to coordinates of the bottom right pixel of the intersection in the destination image.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiCopySubpixIntersect` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function determines the intersection of the source image and the window of size *dstRoiSize* centered in point *point*. Then, corresponding pixels of the destination image are calculated using linear interpolation (see [“Linear Interpolation” on page B-3](#)):

$pDst_{j,i} = pSrc_{Xsubpix(j), Ysubpix(i)}$ , where

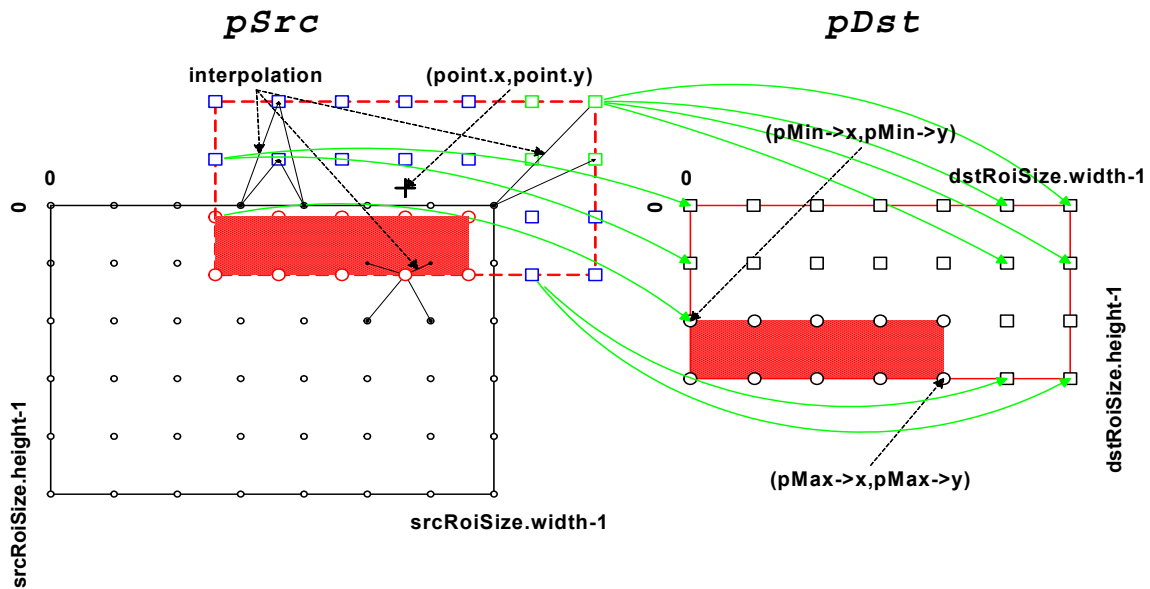
$Xsubpix(j) = \min(\max(point.x + j - 0.5 * (dstRoiSize.width - 1), 0), srcRoiSize.width - 1)$ ,

$Ysubpix(i) = \min(\max(point.y + j - 0.5 * (dstRoiSize.height - 1), 0), srcRoiSize.height - 1)$ ,

$j = 0, \dots, dstRoiSize.width - 1, i = 0, \dots, dstRoiSize.height - 1.$

Minimal values of  $j$  and  $i$  (coordinates of top left calculated destination pixel) are assigned to  $pMin.x$  and  $pMin.y$ , maximal values (coordinates of top left calculated destination pixel) - to  $pMax.x$  and  $pMax.y$ . (See [Figure 4-4](#).)

**Figure 4-4 Image Copying with Subpixel Precision Using `ippiCopySubpixIntersect` Function**



### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoiSize</code> or <code>dstRoiSize</code> has a field with zero or negative value.

<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> is less than <code>srcRoiSize.width * &lt;pixelSize&gt;</code> , or <code>dstStep</code> is less than <code>dstRoiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

---

## Dup

*Copies gray scale image to all channels of the color image.*

---

### Syntax

```
IppStatus, ippIDup_8u_C1C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,  
    int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippIDup` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function copies one-channel (gray scale) image `pSrc` to every channel of the three-channel (color) image `pDst`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

`ippStsSizeErr`Indicates an error condition if *roiSize* has a field with zero or negative value;

## Transpose

*Transpose a source image.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiTranspose_<mod>(const Ipp<datatype>* pSrc, int srcStep,
                               Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R

#### Case 2: In-place operation

```
IppStatus ippiTranspose_<mod>(const Ipp<datatype>* pSrcDst, int
                               srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination ROI for in-place operation.

<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiTranspose` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function transposes the source image *pSrc* (*pSrcDst* for in-place flavors) and stores the result in the *pDst* (*pSrcDst*). The destination image is obtained from the source image by transforming the columns to the rows, that is  
 $pDst(x, y) = pSrc(y, x)$  for each pixel.

The parameter *roiSize* is specified for the source image; therefore the *roiSize.width* for the destination image is equal to the *roiSize.height* for the source image, and the *roiSize.height* for the destination image is equal to the *roiSize.width* for the source image.




---

**CAUTION.** For in-place operations *roiSize.width* must be equal to the *roiSize.height*.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value; or <i>roiSize.width</i> is not equal to <i>roiSize.height</i> for in-place flavors.

## SwapChannels

*Changes the order of channels of the image.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const int dstOrder[3]);
```

Supported values for *mod*:

8u_C3R	16u_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	32s_AC4R	32f_AC4R

#### Case 2: In-place operation

```
ippiSwapChannels_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep,
    IppiSize roiSize, const int dstOrder[3]);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>dstOrder</i>	The order of channels in the destination image.



## Description

The function `ippiSwapChannels` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function changes the order of the channels for each pixel of the source image ROI *pSrc* and stores the swapped values in the destination image ROI *pDst*. First channel in the destination image is determined by the first component of *dstOrder*. Its value lies in the range [0..2] and indicates the corresponding channel number of the source image. Other channels are specified in the similar way. For example, if sequence of channels in source image is *A, B, C* and *dstOrder*[0]=2, *dstOrder*[1]=0, *dstOrder*[2]=1, then the order of channels in the destination image will be *C, A, B*. Some or all components of *dstOrder* may have the same values.

The function does not perform operation on the alpha channel.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of steps values is less than or equal to 0.
<code>ippStsChannelOrderErr</code>	Indicates an error condition if <i>dstOrder</i> is out of the range [0..2].

---

## AddRandUniform\_Direct

*Generates random samples  
with uniform distribution and  
adds them to an image data.*

---

## Syntax

```
IppStatus ippiAddRandUniform_Direct_<mod>(Ipp<datatype>* pSrcDst,  
    int srcDstStep, IppiSize roiSize, Ipp<datatype> low,  
    Ipp<datatype> high, unsigned int* pSeed);
```

Supported values for *mod*:

8u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16s_AC4IR	32f_AC4IR

## Parameters

<i>pSrcDst</i>	Pointer to the source and destination image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>low</i>	The lower bound for the range of uniformly distributed values.
<i>high</i>	The upper bound for the range of uniformly distributed values.
<i>pSeed</i>	The initial seed value for the pseudo-random number generator.

## Description

The function `ippiAddRandUniform_Direct` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function generates samples with uniform distribution over the range `[low, high]` and adds them to a source image pointed to by *pSrcDst*.

The resulting pixel values that exceed the image data range are saturated to the respective data-range limits. To obtain an image which contains pure noise with uniform distribution, call `ippiAddRandUniform_Direct` using a source image with zero data as input.

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

[Example 4-4](#) illustrates the generation of random samples:

---

**Example 4-4 Random Samples Generating**

---

```
IppStatus randUniform( void ) {
    unsigned int seed = 123456;
    Ipp8u img[2048], mn, mx;
    IppiSize roi = { 2048, 1 };
    Ipp64f mean;
    IppStatus st;
    ippiSet_8u_C1R( 0, img, 2048, roi );
    st = ippiAddRandUniform_Direct_8u_C1R( img, 2048, roi, 0, 255, &seed );
    ippiMean_8u_C1R( img, 2048, roi, &mean );
    ippiMinMax_8u_C1R( img, 2048, roi, &mn, &mx );
    printf( "[%d..%d], mean=%.3f\n", mn, mx, mean );
    return st;
}
```

---

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcDstStep</i> has a zero or negative value.

---

## AddRandGauss\_Direct

*Generates random samples with Gaussian distribution and adds them to an image data.*

---

### Syntax

```
IppStatus ippiAddRandGauss_Direct_<mod>(Ipp<datatype>* pSrcDst,  
    int srcDstStep, IppiSize roiSize, Ipp<datatype> mean,  
    Ipp<datatype> stDev, unsigned int* pSeed);
```

Supported values for *mod*:

8u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16s_AC4IR	32f_AC4IR

### Parameters

<i>pSrcDst</i>	Pointer to the source and destination image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>mean</i>	The mean of the Gaussian distribution.
<i>stDev</i>	The standard deviation of the Gaussian distribution.
<i>pSeed</i>	The initial seed value for the pseudo-random number generator.

### Description

The function `ippiAddRandGauss_Direct` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function generates samples with Gaussian distribution that have the mean value *mean* and standard deviation *stdev* and adds them to a source image ROI pointed to by *pSrcDst*.

The resulting pixel values that exceed the image data range are saturated to the respective data-range limits. To obtain an image which contains pure noise with Gaussian distribution, call `ippiAddRandGauss_Direct` using a source image with zero data as input.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrcDst</code> or <code>pSeed</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcDstStep</code> has a zero or negative value.

---

## ImageJaehne

*Creates Jaehne test image.*

---

### Syntax

```
IppStatus ippImageJaehne_<mod>(Ipp<datatype>* pDst, int dstStep,
                                IppiSize roiSize);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>8s_C1R</code>	<code>16u_C1R</code>	<code>16s_C1R</code>	<code>32s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>8s_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32s_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>8s_C4R</code>	<code>16u_C4R</code>	<code>16s_C4R</code>	<code>32s_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>8s_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32s_AC4R</code>	<code>32f_AC4R</code>

### Parameters

<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<code>roiSize</code>	Size of the destination image ROI in pixels.

### Description

The function `ippImageJaehne` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates a specific one- or three-channel test image that has been first introduced to digital image processing by B.Jaehne (see [[Jaehne95](#)]).

The destination image pixel values are computed according to the following formula:

$$\text{Dst}(x, y) = A \cdot \sin(0.5 \cdot \text{IPP\_PI} \cdot (x_2^2 + y_2^2) / \text{roiSize.height}),$$

where

$x, y$  are pixel coordinates varying in the range

$$0 \leq x \leq \text{roiSize.width}-1, \quad 0 \leq y \leq \text{roiSize.height}-1;$$

$\text{IPP\_PI}$  is the library constant that stands for  $\pi$  value;

$$x_2 = (x - \text{roiSize.width} + 1) / 2.0,$$

$$y_2 = (y - \text{roiSize.height} + 1) / 2.0;$$

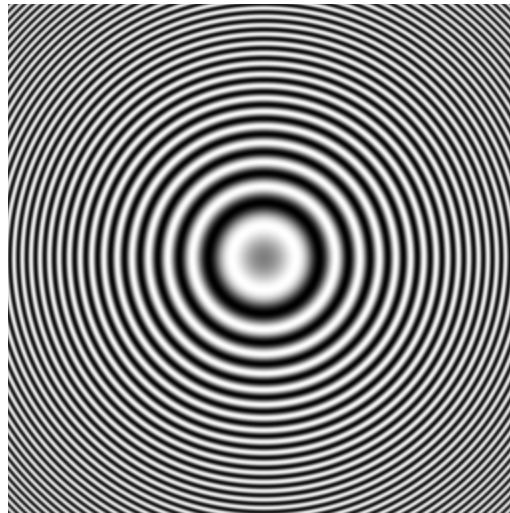
$A$  is the constant value that depends upon the image type being created.

For the 32f floating point data the pixel values in the created image can vary in the range between 0 (inclusive) and 1 (exclusive).

[Figure 4-5](#) illustrates an example of a test image generated by the `ippiImageJaehne` function.

**Figure 4-5 Example of a Generated Jaehne's Test Image**

---



These test images can be effectively used when you need to visualize and interpret the results of applying filtering functions, similarly to what is proposed in [\[Jaehne95\]](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or if <i>DstStep</i> is less than or equal to zero.

## ImageRamp

*Creates a test image  
that has an intensity ramp.*

### Syntax

```
IppStatus ippiImageRamp_<mod>(Ipp<datatype>* pDst, int dstStep,  
                               IppiSize roiSize, float offset, float slope, IppiAxis axis);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>8s_C1R</code>	<code>16u_C1R</code>	<code>16s_C1R</code>	<code>32s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>8s_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32s_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>8s_C4R</code>	<code>16u_C4R</code>	<code>16s_C4R</code>	<code>32s_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>8s_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32s_AC4R</code>	<code>32f_AC4R</code>

### Parameters

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>roiSize</i>	Size of the destination image ROI in pixels.
<i>offset</i>	Offset value.
<i>slope</i>	Slope coefficient.
<i>axis</i>	Specifies the direction of the image intensity ramp; can be one of the following:  <code>ippAxsHorizontal</code> for the ramp in X-direction, <code>ippAxsVertical</code> for the ramp in Y-direction,

`ippAxsBoth`

for ramp in both X and Y-directions.

## Description

The function `ippiImageRamp` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function creates a one- or three-channel image that can be used as a test image to examine the effect of applying different image processing functions.

The destination image pixel values are computed according to one of the following formulas:

$$\begin{aligned} \text{dst}(x,y) &= \text{offset} + \text{slope} * x && , \text{ if } \text{axis} = \text{ippAxsHorizontal}; \\ \text{dst}(x,y) &= \text{offset} + \text{slope} * y && , \text{ if } \text{axis} = \text{ippAxsVertical}; \\ \text{dst}(x,y) &= \text{offset} + \text{slope} * x * y && , \text{ if } \text{axis} = \text{ippAxsBoth}; \end{aligned}$$

where

$x, y$  are pixel coordinates varying in the range  
 $0 \leq x \leq \text{roiSize.width}-1, \quad 0 \leq y \leq \text{roiSize.height}-1$

Note that linear transform coefficients *offset* and *slope* have floating-point values for all function flavors. The computed pixel values that exceed the image data range are saturated to the respective data-range limits.

The code example below illustrates how to use the `ippiImageRamp` function.

### Example 4-5 Creating the Test Image with `ippiImageRamp` Function

---

```
IppStatus ramp( void ) {
    Ipp8u dst[8*4];
    IppiSize roiSize = { 8, 4 };
    return ippiImageRamp_8u_C1R( dst, 8, roiSize, 0.0f, 256.0f/7,
    ippAxsHorizontal );
}
```

The destination image contains the following data:

```
00 25 49 6E 92 B7 DB FF
00 25 49 6E 92 B7 DB FF
00 25 49 6E 92 B7 DB FF
00 25 49 6E 92 B7 DB FF
```

---



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or if <i>DstStep</i> is less than or equal to zero.

---

## SampleLine

*Stores raster line into buffer.*

---

### Syntax

```
IppStatus ippISampleLine_<mod>(const Ipp<DataType>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp<DataType>* pDst, IppiPoint pt1, IppiPoint pt2);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>32f_C3R</code>

### Parameters

<i>pSrc</i>	Pointer to the ROI in the source raster image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the raster image.
<i>roiSize</i>	Size of the image ROI.
<i>pDst</i>	Pointer to the destination buffer. The buffer is to store at least $\max( pt2.x - pt1.x  + 1,  pt2.y - pt1.y  + 1)$ points.
<i>pt1</i>	Starting point of the line.
<i>pt2</i>	Ending point of the line.

### Description

The function `SampleLine` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function iterates through the points that belong to the raster line using the 8-point connected Bresenham algorithm, and stores the resulting pixels in the destination buffer.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize.width</code> or <code>roiSize.height</code> is less than or equal to zero.
<code>ippStsStepErr</code>	Indicates an error when <code>srcStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error when the step for the floating-point image cannot be divided by 4.
<code>ippStsOutOfRangeErr</code>	Indicates an error when any of the line ending points is outside the image.

---

## ZigzagFwd8x8

*Converts a conventional order to the zigzag order.*

---

### Syntax

```
IppStatus ippiZigzagFwd8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

### Parameters

<code>pSrc</code>	Pointer to the source data.
<code>pDst</code>	Pointer to the destination data.

### Description

The function `ippiZigzagFwd8x8` is declared in the `ippi.h` file. This function rearranges data in an 8x8 block from a conventional order (left-to-right, top-to-bottom) to the zigzag sequence.

The zigzag sequence is specified in [Figure 4-6](#).

**Figure 4-6      Zigzag Sequence**

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

**Return Values**

- `ippStsNoErr`Indicates no error.
- `ippStsNullPtrErr`Indicates an error condition if one of the specified pointers are NULL.

**ZigzagInv8x8**

*Converts a zigzag order to the conventional order.*

**Syntax**

```
IppStatus ippZigzagInv8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

**Parameters**

- `pSrc`Pointer to the source data.
- `pDst`Pointer to the destination data.

**Description**

The function `ippZigzagInv8x8` is declared in the `ippi.h` file. This function rearranges data in an 8x8 block from a zigzag order to the conventional order (left-to-right, top-to-bottom). The zigzag sequence is specified in [Figure 4-6](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers are NULL.

# Image Arithmetic and Logical Operations



This chapter describes Intel IPP image processing functions that modify pixel values of an image buffer using arithmetic or logical operations. It also includes functions that perform image compositing based on opacity (alpha-blending).

[Table 5-1](#) lists functions described in more detail later in this chapter:

**Table 5-1      Arithmetic and Logical Functions**

Function Base Name	Operation
<b>Arithmetic Functions</b>	
<a href="#">Add</a>	Adds pixel values of two image buffers
<a href="#">AddC</a>	Adds a constant to pixel values of an image buffer
<a href="#">AddSquare</a>	Adds squared pixel values of a source image to floating-point pixel values of an accumulator image
<a href="#">AddProduct</a>	Adds product of pixel values of two source images to floating-point pixel values of an accumulator image
<a href="#">AddWeighted</a>	Adds pixel values of a source image multiplied by factor $\alpha$ to floating-point pixel values of an accumulator image multiplied by factor $(1 - \alpha)$
<a href="#">Mul</a>	Multiplies pixel values of two image buffers
<a href="#">MulC</a>	Multiplies pixel values of an image buffer by a constant
<a href="#">MulScale</a>	Multiplies pixel values of two image buffers and scales the products
<a href="#">MulCScale</a>	Multiplies pixel values of an image buffer by a constant and scales the products
<a href="#">Sub</a>	Subtracts pixel values of two image buffers
<a href="#">SubC</a>	Subtracts a constant from pixel values of an image buffer
<a href="#">Div</a>	Divides pixel values of two image buffers

**Table 5-1 Arithmetic and Logical Functions (continued)**

Function Base Name	Operation
<a href="#"><u>DivC</u></a>	Divides pixel values of an image buffer by a constant
<a href="#"><u>Abs</u></a>	Computes absolute pixel values of a source image and places them into the destination image
<a href="#"><u>AbsDiff</u></a>	Finds the absolute difference between two images.
<a href="#"><u>AbsDiffC</u></a>	Finds the absolute difference between an image and a scalar value.
<a href="#"><u>Sqr</u></a>	Squares pixel values of an image and writes them into the destination image
<a href="#"><u>Sqrt</u></a>	Computes square roots of pixel values of a source image and writes them into the destination image
<a href="#"><u>Ln</u></a>	Computes the natural logarithm of pixel values in a source image and writes the results into the destination image
<a href="#"><u>Exp</u></a>	Computes the exponential of pixel values in a source image and writes the results into the destination image
<a href="#"><u>Complement</u></a>	Converts negative number from the complement to direct code.
<b>Logical Functions</b>	
<a href="#"><u>And</u></a>	Combines corresponding pixels of two image buffers by a bitwise AND operation.
<a href="#"><u>AndC</u></a>	Performs a bitwise AND operation on each pixel with a constant.
<a href="#"><u>Or</u></a>	Combines corresponding pixels of two image buffers by a bitwise OR operation.
<a href="#"><u>OrC</u></a>	Performs a bitwise OR operation on each pixel with a constant.
<a href="#"><u>Xor</u></a>	Combines corresponding pixels of two image buffers by a bitwise XOR operation.
<a href="#"><u>XorC</u></a>	Performs a bitwise OR operation on each pixel with a constant.
<a href="#"><u>Not</u></a>	Performs a bitwise NOT operation on each pixel
<a href="#"><u>RShiftC</u></a>	Divides pixel values by a constant power of 2 by shifting bits logically or arithmetically to the right.
<a href="#"><u>LShiftC</u></a>	Shifts bits in pixel values to the left.

**Table 5-1 Arithmetic and Logical Functions (continued)**

Function Base Name	Operation
<b>Alpha-Blending Functions</b>	
<a href="#">AlphaComp</a>	Combines two image buffers using alpha (opacity) values.
<a href="#">AlphaCompC</a>	Combines two image buffers using constant alpha values
<a href="#">AlphaPremul</a>	Pre-multiplies pixel values of an image buffer by its alpha values.
<a href="#">AlphaPremulC</a>	Pre-multiplies pixel values of an image buffer using constant alpha values.

An additional suffix C, if present in the function name, indicates operation with a constant. Arithmetic functions that operate on integer data perform fixed scaling of the internally computed results. In case of overflow the result value is saturated to the destination data type range.



**NOTE.** Most arithmetic and logical functions support data with 1-, 3-, or 4-channel pixel values. In the alpha channel case (AC4), the alpha channels are not processed.

Many solutions and hints for use of these functions can be found in Intel IPP Samples. See *Intel IPP Image Processing Samples* downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>.

## Arithmetic Operations

Functions described in this section perform arithmetic operations on pixel values. Arithmetic operations include addition, multiplication, subtraction, and division of pixel values of two images as well as similar operations on a single image and a constant. Computation of absolute value, square, square root, exponential, and natural logarithm of pixels in an image buffer is also supported. Functions of this group perform operations on each pixel in the source buffer(s), and write the results into the destination buffer. Some functions also support processing of images with complex data.

---

### Add

*Adds pixel values of two image buffers.*

---

#### Syntax

##### Case 1: Not-in-place operation on integer or complex data.

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs	16sc_C1RSfs	32sc_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs	16sc_C3RSfs	32sc_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs	16sc_AC4RSfs	32sc_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs		

##### Case 2: Not-in-place operation on floating-point or complex data.

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f_C1R	32fc_C1R
32f_C3R	32fc_C3R
32f_AC4R	32fc_AC4R
32f_C4R	



**Case 3: In-place operation on integer or complex data.**

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
    int scaleFactor);
```

Supported values for *mod*:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs	16sc_C1IRSfs	32sc_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs	16sc_C3IRSfs	32sc_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs	16sc_AC4IRSfs	32sc_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs		

**Case 4: In-place operation on floating-point or complex data.**

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f_C1IR	32fc_C1IR
32f_C3IR	32fc_C3IR
32f_AC4IR	32fc_AC4IR
32f_C4IR	

**Case 5: In-place operation using a floating-point accumulator image.**

```
IppStatus ippiAdd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u32f_C1IR	8s32f_C1IR	16u32f_C1IR
------------	------------	-------------

**Case 6: Masked in-place operation using a floating-point accumulator image.**

```
IppStatus ippiAdd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u32f_C1IMR	8s32f_C1IMR	16u32f_C1IMR	32f_C1IMR
-------------	-------------	--------------	-----------

## Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source image ROIs.
<i>srcStep, src1Step, src2Step</i>	Distance in bytes between starts of consecutive lines in the source image buffers.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>dstSrcStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiAdd` is declared in the `ippi.h` file with the exception of the function flavors in **Case5** and **Case6**, which are declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function adds corresponding pixel values of two source image buffers and places the results in a destination buffer. In case of operations on integer data, the resulting values are scaled by *scaleFactor*. For complex data, the function processes both real and imaginary parts of pixel values.

Some function flavors add 8u, 8s, 16u, or 32f source image pixel values to a floating point accumulator image in-place. Addition of pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.




---

**WARNING.** Step values must be positive for functions that operate on complex data, and no less than `roiSize.width*<pixelSize>` for functions using an accumulator image.

---

Note that the functions with AC4 descriptor do not process alpha channel.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in the following cases: for functions that operate on complex data, if any of the specified step values is zero or negative; for functions using an accumulator image, if any of the specified step values is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

---

## AddC

*Adds a constant to pixel values  
of an image buffer.*

---

### Syntax

#### Case 1: Not-in-place operation on 1-channel integer or complex data.

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C1RSfs      16u\_C1RSfs      16s\_C1RSfs      16sc\_C1RSfs      32sc\_C1RSfs

**Case 2: Not-in-place operation on multi-channel integer or complex data.**

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C3RSfs      16u\_C3RSfs      16s\_C3RSfs      16sc\_C3RSfs      32sc\_C3RSfs  
 8u\_AC4RSfs      16u\_AC4RSfs      16s\_AC4RSfs      16sc\_AC4RSfs      32sc\_AC4RSfs

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C4RSfs      16u\_C4RSfs      16s\_C4RSfs

**Case 3: Not-in-place operation on 1-channel floating-point or complex data.**

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

32f\_C1R      32fc\_C1R

**Case 4: Not-in-place operation on multi-channel floating-point or complex data.**

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

32f\_C3R      32fc\_C3R  
 32f\_AC4R      32fc\_AC4R

```
IppStatus ippiAddC_32f_C4R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

**Case 5: In-place operation on 1-channel integer or complex data.**

```
IppStatus ippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C1IRSfs      16u\_C1IRSfs      16s\_C1IRSfs      16sc\_C1IRSfs      32sc\_C1IRSfs

#### Case 6: In-place operation on multi-channel integer or complex data.

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C3IRSfs      16u\_C3RSfs      16s\_C3IRSfs      16sc\_C3IRSfs      32sc\_C3IRSfs  
8u\_AC4IRSfs      16u\_AC4RSfs      16s\_AC4IRSfs      16sc\_AC4IRSfs      32sc\_AC4IRSfs

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C4IRSfs      16u\_C4IRSfs      16s\_C4IRSfs

#### Case 7: In-place operation on 1-channel floating-point or complex data.

```
IppStatus ippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f\_C1IR      32fc\_C1IR

#### Case 8: In-place operation on multi-channel floating-point or complex data.

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3],
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f\_C3IR      32fc\_C3IR  
32f\_AC4IR      32fc\_AC4IR

```
IppStatus ippiAddC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.

<i>value</i>	The constant value to add to image pixel values (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiAddC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function changes the image intensity by adding the *value* to image pixel values. For one-channel images, a positive *value* brightens the image (increases the intensity); a negative *value* darkens the image (decreases the intensity). For multi-channel images, the components of a constant vector *value* are added to pixel channel values. For complex data, the function processes both real and imaginary parts of pixel values.




---

**WARNING.** Step values must be positive for functions that operate on complex data.

---

In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

Note that the functions with AC4 descriptor do not process alpha channel.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.

---

<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

---

## AddSquare

*Adds squared pixel values of a source image to floating-point pixel values of an accumulator image.*

---

### Syntax

#### Case 1: In-place operation.

```
IppStatus ippAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,  
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u32f_C1IR`      `8s32f_C1IR`      `16u32f_C1IR`      `32f_C1IR`

#### Case 2: Masked in-place operation.

```
IppStatus ippAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep,  
    IppiSize roiSize);
```

Supported values for `mod`:

`8u32f_C1IMR`      `8s32f_C1IMR`      `16u32f_C1IMR`      `32f_C1IMR`

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image buffer.
<code>pMask</code>	Pointer to the mask image.
<code>maskStep</code>	Distance in bytes between starts of consecutive lines in the mask image.
<code>pSrcDst</code>	Pointer to the destination (accumulator) image ROI.

<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the accumulator image.
<i>roiSize</i>	Size of the image ROI in pixels.

## Description

The function `ippiAddSquare` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function adds squared pixel values of the source image *pSrc* to floating-point pixel values of the accumulator image *pSrcDst* as follows:

$$pSrcDst(x, y) = pSrcDst(x, y) + pSrc(x, y)^2$$

Addition of the squared pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize.width</i> or <i>roiSize.height</i> is negative.
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> , <i>maskStep</i> , or <i>srcDstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.



---

## AddProduct

*Adds product of pixel values of two source images to floating-point pixel values of an accumulator image.*

---

### Syntax

#### Case 1: In-place operation.

```
IppStatus ippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1, int
    src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u32f\_C1IR      8s32f\_C1IR      16u32f\_C1IR      32f\_C1IR

#### Case 2: Masked in-place operation.

```
IppStatus ippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1,
    int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step,
    const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u32f\_C1IMR      8s32f\_C1IMR      16u32f\_C1IMR      32f\_C1IMR

### Parameters

<i>pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>pSrcDst</i>	Pointer to the destination (accumulator) image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the accumulator image.
<i>roiSize</i>	Size of the image ROI in pixels.

## Description

The function `ippiAddProduct` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function adds the product of pixel values of two source images `pSrc1` and `pSrc2` to floating-point pixel values of the accumulator image `pSrcDst` as given by:

$$pSrcDst(x, y) = pSrcDst(x, y) + pSrc1(x, y) * pSrc2(x, y)$$

Addition of the products of pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <code>roiSize.width</code> or <code>roiSize.height</code> is negative.
<code>ippStsStepErr</code>	Indicates an error if <code>src1Step</code> , <code>src2Step</code> , <code>maskStep</code> , or <code>srcDstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

---

## AddWeighted

*Adds pixel values of a source image multiplied by factor  $\alpha$  to floating-point pixel values of an accumulator image multiplied by factor  $(1 - \alpha)$ .*

---

## Syntax

### Case 1: In-place operation.

```
ippStatus ippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize,
    Ipp32f alpha);
```

Supported values for *mod*:

8u32f\_C1IR      8s32f\_C1IR      16u32f\_C1IR      32f\_C1IR

## Case 2: Masked in-place operation.

```
IppStatus ippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst,
    int srcDstStep, IppiSize roiSize, Ipp32f alpha);
```

Supported values for *mod*:

8u32f\_C1IMR      8s32f\_C1IMR      16u32f\_C1IMR      32f\_C1IMR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>pSrcDst</i>	Pointer to the destination (accumulator) image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the accumulator image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>alpha</i>	Weight $\alpha$ of the source image.

## Description

The function `ippiAddWeighted` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function adds pixel values of the source image *pSrc* multiplied by a weight factor  $\alpha$  to floating-point pixel values of the accumulator image *pSrcDst* multiplied by  $(1 - \alpha)$  as follows:

$$pSrcDst(x, y) = pSrcDst(x, y) * (1 - \alpha) + pSrc(x, y) * \alpha$$

Addition of the weighted pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize.width</i> or <i>roiSize.height</i> is negative.
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> , <i>maskStep</i> , or <i>srcDstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

---

## Mul

*Multiplies pixel values of two image buffers.*

---

### Syntax

#### Case 1: Not-in-place operation on integer or complex data.

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

<code>8u_C1RSfs</code>	<code>16u_C1RSfs</code>	<code>16s_C1RSfs</code>	<code>16sc_C1RSfs</code>	<code>32sc_C1RSfs</code>
<code>8u_C3RSfs</code>	<code>16u_C3RSfs</code>	<code>16s_C3RSfs</code>	<code>16sc_C3RSfs</code>	<code>32sc_C3RSfs</code>
<code>8u_AC4RSfs</code>	<code>16u_AC4RSfs</code>	<code>16s_AC4RSfs</code>	<code>16sc_AC4RSfs</code>	<code>32sc_AC4RSfs</code>
<code>8u_C4RSfs</code>	<code>16u_C4RSfs</code>	<code>16s_C4RSfs</code>		

#### Case 2: Not-in-place operation on floating-point or complex data.

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int
    dstStep, IppiSize roiSize);
```

Supported values for *mod*:

<code>32f_C1R</code>	<code>32fc_C1R</code>
<code>32f_C3R</code>	<code>32fc_C3R</code>

```
32f_AC4R    32fc_AC4R
32f_C3R
```

### Case 3: In-place operation on integer or complex data.

```
IppStatus ippMul_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
int scaleFactor);
```

Supported values for *mod* :

```
8u_C1IRSfs    16u_C1IRSfs    16s_C1IRSfs    16sc_C1IRSfs    32sc_C1IRSfs
8u_C3IRSfs    16u_C3IRSfs    16s_C3IRSfs    16sc_C3IRSfs    32sc_C3IRSfs
8u_AC4IRSfs    16u_AC4IRSfs    16s_AC4IRSfs    16sc_AC4IRSfs    32sc_AC4IRSfs
8u_C4IRSfs    16u_C4IRSfs    16s_C4IRSfs
```

### Case 4: In-place operation on floating-point or complex data.

```
IppStatus ippMul_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR      32fc_C1IR
32f_C3IR      32fc_C3IR
32f_AC4IR     32fc_AC4IR
32f_C4IR
```

## Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source image buffers.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.

<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiMul` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies corresponding pixel values of two source image buffers and places the results in a destination buffer. In case of operations on integer data, the resulting values are scaled by *scaleFactor*. For complex data, the function processes both real and imaginary parts of pixel values.




---

**WARNING.** Step values must be positive for functions that operate on complex data.

---

Note that the functions with AC4 descriptor do not process alpha channel.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

## MulC

*Multiplies pixel values of an image buffer by a constant.*

### Syntax

#### Case 1: Not-in-place operation on 1-channel integer or complex data.

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C1RSfs      16u\_C1RSfs      16s\_C1RSfs      16sc\_C1RSfs      32sc\_C1RSfs

#### Case 2: Not-in-place operation on multi-channel integer or complex data.

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C3RSfs      16u\_C3RSfs      16s\_C3RSfs      16sc\_C3RSfs      32sc\_C3RSfs  
8u\_AC4RSfs      16u\_AC4RSfs      16s\_AC4RSfs      16sc\_AC4RSfs      32sc\_AC4RSfs

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C4RSfs      16u\_C4RSfs      16s\_C4RSfs

#### Case 3: Not-in-place operation on 1-channel floating-point or complex data.

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod* :

32f\_C1R      32fc\_C1R

**Case 4: Not-in-place operation on multi-channel floating-point or complex data.**

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
    Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C3R      32fc_C3R
32f_AC4R     32fc_AC4R
```

```
IppStatus ippiMulC_32f_C4R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

**Case 5: In-place operation on 1-channel integer or complex data.**

```
IppStatus ippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs   16u_C1IRSfs   16s_C1IRSfs   16sc_C1IRSfs   32sc_C1IRSfs
```

**Case 6: In-place operation on multi-channel integer or complex data.**

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C3IRSfs   16u_C3IRSfs   16s_C3IRSfs   16sc_C3IRSfs   32sc_C3IRSfs
8u_AC4IRSfs   16u_AC4IRSfs   16s_AC4IRSfs   16sc_AC4IRSfs   32sc_AC4IRSfs
```

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C4IRSfs   16u_C4IRSfs   16s_C4IRSfs
```

**Case 7: In-place operation on 1-channel floating-point or complex data.**

```
IppStatus ippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR     32fc_C1IR
```



**Case 8: In-place operation on multi-channel floating-point or complex data.**

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3],
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C3IR      32fc_C3IR
32f_AC4IR     32fc_AC4IR
```

```
IppStatus ippiMulC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>value</i>	The constant value to add to image pixel values (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling.

**Description**

The function `ippiMulC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies pixel values of an image by a constant *value*. For multi-channel images, pixel channel values are multiplied by the components of a constant vector *value*. For complex data, the function processes both real and imaginary parts of pixel values.




---

**WARNING.** Step values must be positive for functions that operate on complex data.

---

In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

Note that the functions with AC4 descriptor do not process alpha channel.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

---

## MulScale

*Multiplies pixel values of two image buffers and scales the products.*

---

### Syntax

#### Case 1: Not-in-place operation .

```
IppStatus ippiMulScale_<mod>(const Ipp<datatype>* pSrc1, int src1Step,  
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,  
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R
8u_C3R	16u_C3R
8u_C4R	16u_C4R
8u_AC4R	16u_AC4R

#### Case 2: In-place operation .

```
IppStatus ippiMulScale_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR
8u_C3IR	16u_C3IR
8u_C4IR	16u_C4IR
8u_AC4IR	16u_AC4IR

### Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source image buffers.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.

<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiMulScale` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies corresponding pixel values of two input buffers and scales the products using the following formula:

$$\text{dst\_pixel} = \text{src1\_pixel} * \text{src2\_pixel} / \text{max\_val},$$

where `src1_pixel` and `src2_pixel` are pixel values of the source buffers, `dst_pixel` is the resultant pixel value, and `max_val` is the maximum value of the pixel data range (see [Table 2-2](#) in Chapter 2 for details).

The function is implemented for 8-bit and 16-bit unsigned data types only.

Note that the functions with AC4 descriptor do not process alpha channel.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

---

## MulCScale

*Multiplies pixel values of an image buffer by a constant and scales the products.*

---

### Syntax

#### Case 1: Not-in-place operation on 1-channel data.

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize
    roiSize);
```

Supported values for *mod*:

8u\_C1R                  16u\_C1R

#### Case 2: Not-in-place operation on multi-channel data.

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u\_C3R                  16u\_C3R  
8u\_AC4R                16u\_AC4R

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u\_C4R                  16u\_C4R

#### Case 3: In-place operation on 1-channel data.

```
IppStatus ippiMulCScale_<mod>(Ipp<datatype> value,
    const Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1IR                16u\_C1IR

#### Case 4: In-place operation on multi-channel data.

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype> value[3], const
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3IR      16u_C3IR
8u_AC4IR     16u_AC4IR
```

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype> value[4], const
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C4IR      16u_C4IR
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>value</i>	The constant value to multiply each pixel value in a source buffer (constant vector in case of 3- or 4-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiMulCScale` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies pixel values in the input buffer by a constant *value* and scales the products using the following formula:

$$\text{dst\_pixel} = \text{src\_pixel} * \text{value} / \text{max\_val},$$

where `src_pixel` is a pixel value in the source buffer, `dst_pixel` is the resultant pixel value, and `max_val` is the maximum value of the pixel data range (see [Table 2-2](#) in Chapter 2 for details).

The function is implemented for 8-bit and 16-bit unsigned data types only. It can be used to multiply pixel values by a number between 0 and 1.

Note that the functions with AC4 descriptor do not process alpha channel.

Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> , <code>pDst</code> , or <code>pSrcDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

Sub

*Subtracts pixel values of two buffers.*

Syntax

Case 1: Not-in-place operation on integer or complex data.

```
IppStatus ippSub<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

<code>8u_C1RSfs</code>	<code>16u_C1RSfs</code>	<code>16s_C1RSfs</code>	<code>16sc_C1RSfs</code>	<code>32sc_C1RSfs</code>
<code>8u_C3RSfs</code>	<code>16u_C3RSfs</code>	<code>16s_C3RSfs</code>	<code>16sc_C3RSfs</code>	<code>32sc_C3RSfs</code>
<code>8u_AC4RSfs</code>	<code>16u_AC4RSfs</code>	<code>16s_AC4RSfs</code>	<code>16sc_AC4RSfs</code>	<code>32sc_AC4RSfs</code>
<code>8u_C4RSfs</code>	<code>16u_C4RSfs</code>	<code>16s_C4RSfs</code>		

**Case 2: Not-in-place operation on floating-point or complex data.**

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1R    32fc_C1R
32f_C3R    32fc_C3R
32f_AC4R   32fc_AC4R
32f_C4R
```

**Case 3: In-place operation on integer or complex data.**

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
    int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs    16u_C1IRSfs    16s_C1IRSfs    16sc_C1IRSfs    32sc_C1IRSfs
8u_C3IRSfs    16u_C3IRSfs    16s_C3IRSfs    16sc_C3IRSfs    32sc_C3IRSfs
8u_AC4IRSfs    16u_AC4IRSfs    16s_AC4IRSfs    16sc_AC4IRSfs    32sc_AC4IRSfs
8u_C4IRSfs    16u_C4IRSfs    16s_C4IRSfs
```

**Case 4: In-place operation on floating-point or complex data.**

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR    32fc_C1IR
32f_C3IR    32fc_C3IR
32f_AC4IR   32fc_AC4IR
32f_C4IR
```

**Parameters**

<i>pSrc</i> , <i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>srcStep</i> , <i>src1Step</i> , <i>src2Step</i>	Distances in bytes between starts of consecutive lines in the source image buffers.
<i>pDst</i>	Pointer to the destination image ROI.



<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiSub` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function subtracts pixel values of the source buffer *pSrc1* from the corresponding pixel values of the buffer *pSrc2* and places the result in a destination buffer *pDst*. For in-place operations, the values in *pSrc* are subtracted from the values in *pSrcDst* and the results are placed into *pSrcDst*. For complex data, the function processes both real and imaginary parts of pixel values.




---

**WARNING.** Step values must be positive for functions that operate on complex data.

---

In case of operations on integer data, the resulting values are scaled by *scaleFactor*. Note that the functions with AC4 descriptor do not process alpha channel.

## Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippiStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippiStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

## SubC

*Subtracts a constant from pixel values of an image buffer.*

---

### Syntax

#### Case 1: Not-in-place operation on 1-channel integer or complex data.

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C1RSfs      16u\_C1RSfs      16s\_C1RSfs      16sc\_C1RSfs      32sc\_C1RSfs

#### Case 2: Not-in-place operation on multi-channel integer or complex data.

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C3RSfs      16u\_C3RSfs      16s\_C3RSfs      16sc\_C3RSfs      32sc\_C3RSfs  
8u\_AC4RSfs      16u\_AC4RSfs      16s\_AC4RSfs      16sc\_AC4RSfs      32sc\_AC4RSfs

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C4RSfs      16u\_C4RSfs      16s\_C4RSfs

#### Case 3: Not-in-place operation on 1-channel floating-point or complex data.

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

32f\_C1R      32fc\_C1R

**Case 4: Not-in-place operation on multi-channel floating-point or complex data.**

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
    Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize
    roiSize);
```

Supported values for *mod*:

```
32f_C3R      32fc_C3R
32f_AC4R     32fc_AC4R
```

```
IppStatus ippiSubC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f
    value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

**Case 5: In-place operation on 1-channel integer or complex data.**

```
IppStatus ippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs   16u_C1IRSfs   16s_C1IRSfs   16sc_C1IRSfs   32sc_C1IRSfs
```

**Case 6: In-place operation on multi-channel integer or complex data.**

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C3IRSfs   16u_C3IRSfs   16s_C3IRSfs   16sc_C3IRSfs   32sc_C3IRSfs
8u_AC4IRSfs   16u_AC4IRSfs   16s_AC4IRSfs   16sc_AC4IRSfs   32sc_AC4IRSfs
```

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C4IRSfs   16u_C4IRSfs   16s_C4IRSfs
```

**Case 7: In-place operation on 1-channel floating-point or complex data.**

```
IppStatus ippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR     32fc_C1IR
```

### Case 8: In-place operation on multi-channel floating-point or complex data.

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3],
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C3IR      32fc_C3IR
32f_AC4IR     32fc_AC4IR
```

```
IppStatus ippiSubC_32f_C4IR(const Ipp32f> value[4], Ipp32f* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>value</i>	The constant value to subtract from each pixel value in a source buffer (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling.

### Description

The function `ippiSubC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function changes image intensity by subtracting the constant *value* from pixel values of an image buffer. For multi-channel images, the components of a constant vector *value* are subtracted from pixel channel values. For complex data, the function processes both real and imaginary parts of pixel values.



---

**WARNING.** Step values must be positive for functions that operate on complex data.

---

In case of operations on integer data, the resulting values are scaled by *scaleFactor*.

Note that the functions with AC4 descriptor do not process alpha channel.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

## Div

*Divides pixel values of an image buffer by pixel values of another buffer.*

---

### Syntax

#### Case 1: Not-in-place operation on integer or complex data.

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16s_C1RSfs	16sc_C1RSfs	32sc_C1RSfs
8u_C3RSfs	16s_C3RSfs	16sc_C3RSfs	32sc_C3RSfs
8u_AC4RSfs	16s_AC4RSfs	16sc_AC4RSfs	32sc_AC4RSfs
8u_C4RSfs	16s_C4RSfs		

#### Case 2: Not-in-place operation on floating-point or complex data.

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int
    dstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f_C1R	32fc_C1R
32f_C3R	32fc_C3R
32f_AC4R	32fc_AC4R
32f_C4R	

#### Case 3: In-place operation on integer or complex data.

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
    int scaleFactor);
```

Supported values for *mod*:

8u_C1IRSfs	16s_C1IRSfs	16sc_C1IRSfs	32sc_C1IRSfs
8u_C3IRSfs	16s_C3IRSfs	16sc_C3IRSfs	32sc_C3IRSfs
8u_AC4IRSfs	16s_AC4IRSfs	16sc_AC4IRSfs	32sc_AC4IRSfs

```
8u_C4IRSfs 16s_C4IRSfs
```

#### Case 4: In-place operation on floating-point or complex data.

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR      32fc_C1IR
32f_C3IR      32fc_C3IR
32f_AC4IR     32fc_AC4IR
32f_C4IR
```

#### Parameters

<i>pSrc</i> , <i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>srcStep</i> , <i>src1Step</i> , <i>src2Step</i>	Distances in bytes between starts of consecutive lines in the source image buffers.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling.

#### Description

The function `ippiDiv` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function divides pixel values of the source buffer *pSrc2* by the corresponding pixel values of the buffer *pSrc1* and places the result in a destination buffer *pDst*. For in-place operations, the values in *pSrcDst* are divided by the values in *pSrc* and placed into *pSrcDst*. For complex data, the function processes both real and imaginary parts of pixel values. In case of operations on integer data, the resulting values are scaled by *scaleFactor* and rounded (not truncated).

When the function encounters a zero divisor value, the execution is not interrupted. The function returns the warning message and corresponding result value (see appendix A [“Handling of Special Cases”](#) for more information).

Note that the functions with AC4 descriptor do not process alpha channel.

The following code example illustrates the operation of the `ippiDiv` function:

---

#### Example 5-1 Division of Pixel Values

---

```
IppStatus div32f( void ) {
    Ipp32f a[4*3], b[4*3];
    IppiSize roi = {2,2};
    int i;
    for( i=0; i<4*3; ++i ) a[i] = b[i] = (float)i;
    return ippiDiv_32f_C1IR( a, 4*sizeof(Ipp32f), b,
                            4*sizeof(Ipp32f), roi );
}
```

The destination image `b` contains

```
-1.#IND  +1.000  +2.000  +3.000
+1.000   +1.000  +6.000  +7.000
+8.000   +9.000  +10.00 +11.00
```

Console output:

```
-- warning in div32f: (6) Zero value(s) of divisor in the function Div
```

---

#### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.
<code>ippStsDivByZero</code>	Indicates a warning that a divisor value is zero. The function execution is continued.



## DivC

*Divides pixel values of an image buffer  
by a constant.*

### Syntax

#### Case 1: Not-in-place operation on 1-channel integer or complex data.

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C1RSfs      16s\_C1RSfs      16sc\_C1RSfs      32sc\_C1RSfs

#### Case 2: Not-in-place operation on multi-channel integer or complex data.

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C3RSfs      16s\_C3RSfs      16sc\_C3RSfs      32sc\_C3RSfs  
8u\_AC4RSfs      16s\_AC4RSfs      16sc\_AC4RSfs      32sc\_AC4RSfs

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u\_C4RSfs      16s\_C4RSfs

#### Case 3: Not-in-place operation on 1-channel floating-point or complex data.

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

32f\_C1R      32fc\_C1R

**Case 4: Not-in-place operation on multi-channel floating-point or complex data.**

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
    Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize
    roiSize);
```

Supported values for *mod*:

```
32f_C3R      32fc_C3R
32f_AC4R     32fc_AC4R
```

```
IppStatus ippiDivC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f
    val[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

**Case 5: In-place operation on 1-channel integer or complex data.**

```
IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs    16s_C1IRSfs    16sc_C1IRSfs    32sc_C1IRSfs
```

**Case 6: In-place operation on multi-channel integer or complex data.**

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C3IRSfs    16s_C3IRSfs    16sc_C3IRSfs    32sc_C3IRSfs
8u_AC4IRSfs    16s_AC4IRSfs    16sc_AC4IRSfs    32sc_AC4IRSfs
```

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C4IRSfs    16s_C4IRSfs
```

**Case 7: In-place operation on 1-channel floating-point or complex data.**

```
IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR      32fc_C1IR
```

**Case 8: In-place operation on multi-channel floating-point or complex data.**

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3],
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C3IR      32fc_C3IR
32f_AC4IR     32fc_AC4IR
```

```
IppStatus ippiDivC_32f_C4IR(const Ipp32f val[4], Ipp32f* pSrcDst, int
    srcDstStep, IppiSize roiSize);
```

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>value</i>	The constant value to divide each pixel value in a source buffer (constant vector in case of 3- or 4-channel images).
<i>pDst</i>	Pointer to the destination buffer.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling.

**Description**

The function `ippiDivC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function changes image intensity by dividing pixel values of an image buffer by the constant *value*. For multi-channel images, pixel channel values are divided by the components of a constant vector *value*. For complex data, the function processes both real and imaginary parts of pixel values. In case of operations on integer data, the resulting values are scaled by *scaleFactor* and rounded (not truncated).

When the divisor value is zero, the function execution is aborted and the `ippStsDivByZeroErr` error status is set.

Note that in the alpha channel case (AC4), the alpha channels are not processed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.
<code>ippStsDivByZeroErr</code>	Indicates an error condition if the divisor value is zero.

---

## Abs

*Computes absolute pixel values of a source image and places them into the destination image.*

---

### Syntax

#### Case 1: Not-in-place operation .

```
IppStatus ippiAbs_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

<code>16s_C1R</code>	<code>32f_C1R</code>
<code>16s_C3R</code>	<code>32f_C3R</code>
<code>16s_C4R</code>	<code>32f_C4R</code>
<code>16s_AC4R</code>	<code>32f_AC4R</code>

#### Case 2: In-place operation .

```
IppStatus ippiAbs_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

16s_C1IR	32f_C1IR
16s_C3IR	32f_C3IR
16s_C4IR	32f_C4IR
16s_AC4IR	32f_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiAbs` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function takes the absolute value of each channel in each pixel of the source image ROI and places the result into a destination image ROI. It operates on signed data only. Note that the functions with AC4 descriptor do not process alpha channel.

## Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippiStsSizeErr</code>	Indicates an error condition if the <i>roiSize</i> has a field with zero or negative value.

## AbsDiff

*Calculates absolute difference between two images.*

---

### Syntax

```
IppStatus ippAbsDiff_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1R      16u\_C1R      32f\_C1R

### Parameters

<i>pSrc1</i>	Pointer to the first source image.
<i>src1Step</i>	Step in bytes in the first source image.
<i>pSrc2</i>	Pointer to second source image.
<i>src2Step</i>	Step in bytes in the second source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Step in bytes in the destination image.
<i>roiSize</i>	Size of the image ROI in pixels.

### Description

The function `ippAbsDiff` is declared in the `ippi.cv` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the absolute pixel-wise difference between two images by the formula:

$$pDst(x, y) = \text{abs}(pSrc1(x, y) - pSrc2(x, y)) .$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>roiSize</i> has a field with zero or negative value.

---

<code>ippStsStepErr</code>	Indicates an error when <code>src1Step</code> , <code>src2Step</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

---

## AbsDiffC

*Calculates absolute difference between image and scalar value.*

---

### Syntax

```
IppStatus ippAbsDiffC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
                             Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int value);
```

Supported values for *mod*:

```
8u_C1R      16u_C1R
```

```
IppStatus ippAbsDiffC_32f_C1R(const Ipp32f* pSrc, int srcStep,
                               Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f value);
```

### Parameters

<code>pSrc</code>	Pointer to the source image.
<code>srcStep</code>	Step in bytes in the source image.
<code>pDst</code>	Pointer to the destination image.
<code>dstStep</code>	Step in bytes in the destination image.
<code>roiSize</code>	Size of the image ROI in pixels.
<code>value</code>	Scalar value used to decrement each element of the source image.

### Description

The function `ippAbsDiffC` is declared in the `ippi.cv` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the absolute pixel-wise difference between the source image `pSrc` and the scalar `value` by the formula:

$pDst(x, y) = \text{abs}(pSrc(x, y) - \text{value})$ .

For 8u data type the function clips the *value* to the range [0, 255].

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> or <i>dstStep</i> is less than $roiSize.width * \langle pixelSize \rangle$ .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

---

## Sqr

*Squares pixel values of an image  
and writes them into the destination image.*

---

### Syntax

#### Case 1: Not-in-place operation on integer data.

```
IppStatus ippISqr_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

<code>8u_C1RSfs</code>	<code>16u_C1RSfs</code>	<code>16s_C1RSfs</code>
<code>8u_C3RSfs</code>	<code>16u_C3RSfs</code>	<code>16s_C3RSfs</code>
<code>8u_C4RSfs</code>	<code>16u_C4RSfs</code>	<code>16s_C4RSfs</code>
<code>8u_AC4RSfs</code>	<code>16u_AC4RSfs</code>	<code>16s_AC4RSfs</code>

#### Case 2: Not-in-place operation on floating-point data.

```
IppStatus ippISqr_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
```



Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

### Case 3: In-place operation on integer data.

```
IppStatus ippISqr_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1IRSfs    16u_C1IRSfs    16s_C1IRSfs
8u_C3IRSfs    16u_C3IRSfs    16s_C3IRSfs
8u_C4IRSfs    16u_C4IRSfs    16s_C4IRSfs
8u_AC4IRSfs    16u_AC4IRSfs    16s_AC4IRSfs
```

### Case 4: In-place operation on floating-point data.

```
IppStatus ippISqr_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.

<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiSqr` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function squares pixel values of the source image ROI and writes them to the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI.

Note that the functions with AC4 descriptor do not process alpha channel.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>roiSize</i> has a field with zero or negative value.

---

## Sqrt

*Computes square roots of pixel values of a source image and writes them into the destination image.*

---

## Syntax

### Case 1: Not-in-place operation on integer data.

```
IppStatus ippiSqrt_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs

### Case 2: Not-in-place operation on floating-point data.

```
IppStatus ippiSqrt_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f\_C1R  
32f\_C3R  
32f\_AC4R

### Case 3: In-place operation on integer data.

```
IppStatus ippiSqrt_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs

### Case 4: In-place operation on floating-point data.

```
IppStatus ippiSqrt_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

32f\_C1IR  
32f\_C3IR  
32f\_C4IR  
32f\_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling.

### Description

The function `ippiSqrt` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes square roots of pixel values of the source image ROI and writes them into the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI.

If a source pixel value is negative, the function issues a warning and continues execution with the corresponding result value (see appendix A [“Handling of Special Cases”](#) for more information).

Note that the functions with AC4 descriptor do not process alpha channel.

### Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippiStsSizeErr</code>	Indicates an error condition if the <i>roiSize</i> has a field with zero or negative value.
<code>ippiStsSqrtNegArg</code>	Indicates a warning that a source pixel has a negative value.

---

## Ln

*Computes the natural logarithm of pixel values in a source image and writes the results into the destination image.*

---

### Syntax

#### Case 1: Not-in-place operation on integer data.

```
IppStatus ippiLn_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16s_C3RSfs

#### Case 2: Not-in-place operation on floating-point data.

```
IppStatus ippiLn_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f_C1R
32f_C3R

#### Case 3: In-place operation on integer data.

```
IppStatus ippiLn_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16s_C3IRSfs

#### Case 4: In-place operation on floating-point data.

```
IppStatus ippiLn_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
```

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling.

#### Description

The function `ippiLn` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes natural logarithms of pixel values of the source image ROI and writes the resultant values to the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI.

If a source pixel value is zero or negative, the function issues a corresponding warning and continues execution with the corresponding result value (see appendix A “[Handling of Special Cases](#)” for more information).

When several inputs have zero or negative value, the status code returned by the function corresponds to the first encountered case as illustrated in the code [Example 5-2](#):

**Example 5-2 Using the Logarithm Function**

```

IppStatus ln( void ) {
    Ipp32f img[8*8];
    IppiSize roi = { 8, 8 };
    IppStatus st;
    ippiSet_32f_C1R( (float)IPP_E, img, 8*4, roi );
    img[0] = -0;
    img[1] = -1;
    st = ippiLn_32f_C1IR( img, 8*sizeof(Ipp32f), roi );
    printf( "%f %f %f\n", img[0], img[1], img[2] );
    return st;
}

```

Output values:

```
-1.#INF00 -1.#IND00 1.000000
```

Status value and message:

```
(7) Zero value(s) of argument in the Ln function
```

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.
<code>ippStsLnZeroArg</code>	Indicates a warning that a source pixel has a zero value.
<code>ippStsLnNegArg</code>	Indicates a warning that a source pixel has a negative value.

## Exp

*Computes the exponential of pixel values in a source image and writes the results into the destination image.*

---

### Syntax

#### Case 1: Not-in-place operation on integer data.

```
IppStatus ippiExp_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16s_C3RSfs

#### Case 2: Not-in-place operation on floating-point data.

```
IppStatus ippiExp_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

32f_C1R
32f_C3R

#### Case 3: In-place operation on integer data.

```
IppStatus ippiExp_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

8u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16s_C3IRSfs



**Case 4: In-place operation on floating-point data.**

```
IppStatus ippiExp_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
```

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling.

**Description**

The function `ippiExp` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes  $e$  to the power of pixel values of the source image ROI and writes the resultant values into the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by *scaleFactor* to the internally computed values, and saturate the results before writing them to the destination image ROI.

When the overflow occurs, the resultant value is determined in accordance with the data type (see appendix A [“Handling of Special Cases”](#) for more information).

**Return Values**

`ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.

<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

---

## Complement

*Converts negative number from the complement to direct code.*

---

### Syntax

```
IppStatus ippiComplement_32s_C1IR(Ipp32s* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

### Parameters

<i>pSrcDst</i>	Pointer to the source and destination image ROI.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiComplement_32s_C1IR` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts negative integer number from the complement to direct code reserving the sign in the most significant bit.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrcDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcDstStep</i> has a zero or negative value.

`ippStsStrideErr` Indicates an error condition if `srcDstStep` is less than the image width.

## Logical Operations

Functions described in this section perform bitwise operations on pixel values. The operations include logical AND, NOT, inclusive OR, exclusive OR, and bit shifts.

---

### And

*Performs a bitwise AND operation between corresponding pixels of two source buffers.*

---

#### Syntax

##### Case 1: Not-in-place operation .

```
IppStatus ippAnd_<mod>(const Ipp<datatype>* pSrc1, int src1Step,  
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,  
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

##### Case 2: In-place operation .

```
IppStatus ippAnd_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

## Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source image buffers.
<i>pDst</i>	Pointer to the destination image ROI
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiAnd` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise AND operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI.

Note that the functions with AC4 descriptor do not process alpha channel.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## AndC

*Performs a bitwise AND operation of each pixel with a constant.*

### Syntax

#### Case 1: Not-in-place operation on 1-channel data.

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32s_C1R
--------	---------	---------

#### Case 2: Not-in-place operation on multi-channel data.

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	32s_C3R
8u_AC4R	16u_AC4R	32s_AC4R

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C4R	16u_C4R	32s_C4R
--------	---------	---------

#### Case 3: In-place operation on 1-channel data.

```
IppStatus ippiAndC_<mod>(Ipp<datatype> value, const Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	32s_C1IR
---------	----------	----------

#### Case 4: In-place operation on multi-channel data.

```
IppStatus ippiAndC_<mod>(const Ipp<datatype> value[3], const
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3IR	16u_C3IR	32s_C3IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

```
IppStatus ippiAndC_<mod>(const Ipp<datatype> value[4], const
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_AC4IR	16u_AC4IR	32s_AC4IR
----------	-----------	-----------

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>value</i>	The constant value to perform the bitwise AND operation on each pixel of the source image ROI (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

#### Description

The function `ippiAndC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise AND operation between each pixel value of a source image ROI and constant *value*.

Note that the functions with AC4 descriptor do not process alpha channel.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

---

## Not

*Performs a bitwise NOT operation on each pixel of a source buffer.*

---

### Syntax

#### Case 1: Not-in-place operation .

```
IppStatus ippNot_<mod>(const Ipp8u* pSrc, int srcStep,  
                        Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1R  
8u_C3R  
8u_C4R  
8u_AC4R
```

#### Case 2: In-place operation .

```
IppStatus ippNot_<mod>(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C1IR  
8u_C3IR  
8u_C4IR
```

8u\_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiNot` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise NOT operation on each pixel value of a source image ROI.

Note that the functions with AC4 descriptor do not process alpha channel.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.



# Or

Performs bitwise inclusive OR operation between pixels of two source buffers.

## Syntax

### Case 1: Not-in-place operation .

```
IppStatus ippiOr_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

### Case 2: In-place operation .

```
IppStatus ippiOr_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

## Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source images ROI.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source image buffers.
<i>pDst</i>	Pointer to the destination image ROI
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.

<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiOr` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise inclusive OR operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI. Note that the functions with AC4 descriptor do not process alpha channel.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

---

## OrC

*Performs a bitwise inclusive OR operation between each pixel of a buffer and a constant.*

---

## Syntax

### Case 1: Not-in-place operation on 1-channel data.

```

IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);

```

Supported values for *mod*:

8u\_C1R                  16u\_C1R                  32s\_C1R

### Case 2: Not-in-place operation on multi-channel data.

```
IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u\_C3R                  16u\_C3R                  32s\_C3R  
8u\_AC4R                  16u\_AC4R                  32s\_AC4R

```
IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u\_C4R                  16u\_C4R                  32s\_C4R

### Case 3: In-place operation on 1-channel data.

```
IppStatus ippiOrC_<mod>(Ipp<datatype> value,
    const Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1IR                  16u\_C1IR                  32s\_C1IR

### Case 4: In-place operation on multi-channel data.

```
IppStatus ippiOrC_<mod>(const Ipp<datatype> value[3],
    const Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C3IR                  16u\_C3IR                  32s\_C3IR  
8u\_AC4IR                  16u\_AC4IR                  32s\_AC4IR

```
IppStatus ippiOrC_<mod>(const Ipp<datatype> value[4],
    const Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C4IR                  16u\_C4IR                  32s\_C4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>value</i>	The constant value to perform the bitwise OR operation on each pixel of the source buffer (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiOrC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise inclusive OR operation between each pixel value of a source image ROI and constant *value*.

Note that the functions with AC4 descriptor do not process alpha channel.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Xor

*Performs bitwise exclusive OR operation between pixels of two source buffers.*

---

### Syntax

**Case 1: Not-in-place operation .**

```
IppStatus ippiXor_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

**Case 2: In-place operation .**

```
IppStatus ippiXor_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

### Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the source image ROIs.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source image buffers.
<i>pDst</i>	Pointer to the destination image ROI
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.

<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiXor` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise exclusive OR operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI.

Note that the functions with AC4 descriptor do not process alpha channel.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## XorC

*Performs a bitwise exclusive OR operation between each pixel of a buffer and a constant.*

### Syntax

#### Case 1: Not-in-place operation on 1-channel data.

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32s_C1R
--------	---------	---------

#### Case 2: Not-in-place operation on multi-channel data.

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	32s_C3R
8u_AC4R	16u_AC4R	32s_AC4R

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C4R	16u_C4R	32s_C4R
--------	---------	---------

#### Case 3: In-place operation on 1-channel data.

```
IppStatus ippiXorC_<mod>(Ipp<datatype> value,
    const Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	32s_C1IR
---------	----------	----------

#### Case 4: In-place operation on multi-channel data.

```
IppStatus ippiXorC_<mod>(const Ipp<datatype> value[3], const
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3IR	16u_C3IR	32s_C3IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

```
IppStatus ippiXorC_<mod>(const Ipp<datatype> value[4], const
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C4IR	16u_C4IR	32s_C4IR
---------	----------	----------

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>value</i>	The constant value to perform the bitwise XOR operation on each pixel of the source image ROI (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

#### Description

The function `ippiXorC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a bitwise exclusive OR operation between each pixel value of a source image ROI and constant *value*.



Note that the functions with AC4 descriptor do not process alpha channel.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

---

## LShiftC

*Shifts bits in pixel values to the left.*

---

### Syntax

#### Case 1: Not-in-place operation on 1-channel data.

```
IppStatus ippILShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp32u value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

`8u_C1R`      `16u_C1R`      `32s_C1R`

#### Case 2: Not-in-place operation on multi-channel data.

```
IppStatus ippILShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp32u value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

`8u_C3R`      `16u_C3R`      `32s_C3R`  
`8u_AC4R`      `16u_AC4R`      `32s_AC4R`

```
IppStatus ippILShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp32u value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u\_C4R          16u\_C4R          32s\_C4R

### Case 3: In-place operation on 1-channel data.

```
IppStatus ippiLShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1IR          16u\_C1IR          32s\_C1IR

### Case 4: In-place operation on multi-channel data.

```
IppStatus ippiLShiftC_<mod>(const Ipp32u value[3], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C3IR          16u\_C3IR          32s\_C3IR  
8u\_AC4IR          16u\_AC4IR          32s\_AC4IR

```
IppStatus ippiLShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C4IR          16u\_C4IR          32s\_C4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>value</i>	The number of bits to shift (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.

*roiSize*                      Size of the source and destination ROI in pixels.

## Description

The function `ippiLShiftC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function changes the intensity of pixels in the source image ROI by shifting the bits in each pixel value by *value* bits to the left. In case of multi-channel data, each color channel can have its own shift value. The positions vacated after shifting the bits are filled with zeros. Values obtained as a result of left shift operations are not saturated. To get saturated values, use multiplication functions instead.

Note that the functions with AC4 descriptor do not process alpha channel.

The code example below illustrates the use of left shift function.

### Example 5-3    Shifting Bits to the Left

```
IppStatus lshift( void ) {
    Ipp8u img[8*8] = { 1, 0x7F, 0xFE };
    IppiSize roi = { 8, 8 };
    IppStatus st = ippiLShiftC_8u_C1IR( 1, img, 8, roi );
    printf( "%02x %02x %02x\n", img[0], img[1], img[2] );
    return st;
}
```

Output values:

02 fe fc

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## RShiftC

*Shifts bits in pixel values to the right.*

---

### Syntax

#### Case 1: Not-in-place operation on 1-channel data.

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp32u value, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	8s_C1R	16u_C1R	16s_C1R	32s_C1R
--------	--------	---------	---------	---------

#### Case 2: Not-in-place operation on multi-channel data.

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp32u value[3], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	8s_C3R	16u_C3R	16s_C3R	32s_C3R
8u_AC4R	8s_AC4R	16u_AC4R	16s_AC4R	32s_AC4R

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp32u value[4], Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C4R	8s_C4R	16u_C4R	16s_C4R	32s_C4R
--------	--------	---------	---------	---------

#### Case 3: In-place operation on 1-channel data.

```
IppStatus ippiRShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	8s_C1IR	16u_C1IR	16s_C1IR	32s_C1IR
---------	---------	----------	----------	----------

#### Case 4: In-place operation on multi-channel data.

```
IppStatus ippiRShiftC_<mod>(const Ipp32u value[3], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3IR	8s_C3IR	16u_C3IR	16s_C3IR	32s_C3IR
8u_AC4IR	8s_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR

```
IppStatus ippiRShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>*
    pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C4IR	8s_C4IR	16u_C4IR	16s_C4IR	32s_C4IR
---------	---------	----------	----------	----------

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>value</i>	The number of bits to shift (constant vector in case of multi-channel images).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRShiftC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function decreases the intensity of pixels in the source image ROI by shifting the bits in each pixel value by *value* bits to the right. The positions vacated after shifting the bits are filled with the sign bit. In case of multi-channel data, each color channel can have its own shift value. This operation is equivalent to dividing the pixel values by a constant power of 2.

Note that the functions with AC4 descriptor do not process alpha channel.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Alpha Composition

The Intel IPP provides functions that composite two image buffers using either the opacity (alpha) channel in the images or provided alpha values.

These functions operate on image buffers with 8-bit or 16-bit data in RGB or RGBA format. In all compositing operations a resultant pixel in destination buffer *pDst* is created by overlaying a pixel from the foreground image buffer *pSrc1* over a pixel from the background image buffer *pSrc2*. The supported types of images' combining by using alpha values are listed in [Table 5-3](#).

**Table 5-2**      **Types of Image Compositing Operations**

Type	Output Pixel		Description in Imaging Terms
	Color Components	Alpha value	
OVER	$\alpha_A * A + (1 - \alpha_A) * \alpha_B * B$	$\alpha_A + (1 - \alpha_A) * \alpha_B$	A occludes B
IN	$\alpha_A * A * \alpha_B$	$\alpha_A * \alpha_B$	A within B. A acts as a matte for B. A shows only where B is visible.
OUT	$\alpha_A * A * (1 - \alpha_B)$	$\alpha_A * (1 - \alpha_B)$	A outside B. NOT-B acts as a matte for A. A shows only where B is not visible.
ATOP	$\alpha_A * A * \alpha_B + (1 - \alpha_A) * \alpha_B * B$	$\alpha_A * \alpha_B + (1 - \alpha_A) * \alpha_B$	Combination of (A IN B) and (B OUT A). B is both back-ground and matte for A.
XOR	$\alpha_A * A * (1 - \alpha_B) + (1 - \alpha_A) * \alpha_B * B$	$\alpha_A * (1 - \alpha_B) + (1 - \alpha_A) * \alpha_B$	Combination of (A OUT B) and (B OUT A). A and B mutually exclude each other.
PLUS	$\alpha_A * A + \alpha_B * B$	$\alpha_A + \alpha_B$	Blend without precedence

In the formulas above, we denote for simplicity the input image buffers as A and B. The Greek letter  $\alpha$  with subscripts denotes the normalized (scaled) alpha value in the range 0 to 1. It is related to the integer alpha value *alpha* as:

$$\alpha = \text{alpha} / \text{max\_val}$$

where *max\_val* is 255 for 8-bit or 65535 for 16-bit unsigned pixel data.

For the `ippiAlphaComp` function that operates on 4-channel RGBA buffers only,  $\alpha_A$  and  $\alpha_B$  are the normalized alpha values of the two input image buffers, respectively.

For the `ippiAlphaCompC` function,  $\alpha_A$  and  $\alpha_B$  are the normalized constant alpha values that are

passed as parameters to the function.

Note that in formulas for computing the resultant color channel values, A and B stand for the pixel color components of the respective input image buffers.

To save a significant amount of computation for some of the alpha compositing operations, use functions `ippiAlphaPremul`, `ippiAlphaPremulC` for pre-multiplying color channel values by the alpha values. This reduces the number of multiplications required in the compositing operations, which is especially efficient for a repeated compositing of an image.

The type of composition operation to be used by `ippiAlphaComp` and `ippiAlphaCompC` functions is indicated by the `alphaType` parameter, which can take on values listed in the following table:

**Table 5-3      Admissible Values of the `alphaType` Parameter**

Operation Type		Parameter Value
OVER	<code>ippAlphaOver</code>	<code>ippAlphaOverPremul</code>
IN	<code>ippAlphaIn</code>	<code>ippAlphaInPremul</code>
OUT	<code>ippAlphaOut</code>	<code>ippAlphaOutPremul</code>
ATOP	<code>ippAlphaATop</code>	<code>ippAlphaATopPremul</code>
XOR	<code>ippAlphaXor</code>	<code>ippAlphaXorPremul</code>
PLUS	<code>ippAlphaPlus</code>	<code>ippAlphaPlusPremul</code>

## AlphaComp

*Combines two images using alpha (opacity) values of both images.*

### Syntax

```
IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst,
    int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for `mod`:

<code>8u_AC1R</code>	<code>8u_AC4R</code>
<code>16u_AC1R</code>	<code>16u_AC4R</code>



```
IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* const pSrc1[4],
    int src1Step, const Ipp<datatype>* const pSrc2[4], int src2Step,
    Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize,
    IppiAlphaType alphaType);
```

Supported values for *mod*:

8u\_AP4R      16u\_AP4R

## Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
<i>src1Step</i> , <i>src2Step</i>	Distances in bytes between starts of consecutive lines in the source image buffers.
<i>pDst</i>	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>alphaType</i>	The composition type to perform. See <a href="#">Table 5-3</a> for the type value and description.

## Description

The function `ippiAlphaComp` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs an image compositing operation on RGBA images using alpha values of both images. The compositing is done by overlaying pixels  $(r_A, g_A, b_A, \alpha_A)$  from the foreground image *pSrc1* with pixels  $(r_B, g_B, b_B, \alpha_B)$  from the background image *pSrc2* to produce pixels  $(r_C, g_C, b_C, \alpha_C)$  in the resultant image *pDst*.

The alpha values are assumed to be normalized to the range [0..1].

The type of the compositing operation is indicated by the *alphaType* parameter.

Use [Table 5-3](#) to choose a valid *alphaType* value depending on the required composition type.

For example, the resulting pixel color components for the OVER operation (see [Table 5-2](#)) are computed as follows:

$$r_C = \alpha_A * r_A + (1 - \alpha_A) * \alpha_B * r_B$$

$$g_C = \alpha_A * g_A + (1 - \alpha_A) * \alpha_B * g_B$$

$$b_C = \alpha_A * b_A + (1 - \alpha_A) * \alpha_B * b_B$$

The resulting (normalized) alpha value is computed as

$$\alpha_C = \alpha_A + (1 - \alpha_A) * \alpha_B$$

This function can be used for unsigned pixel data only.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <code>roiSize</code> has a field with zero or negative value.

---

## AlphaCompC

*Combines two images using constant alpha values.*

---

### Syntax

```
IppStatus ippAlphaCompC_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    Ipp<datatype> alpha1, const Ipp<datatype>* pSrc2, int src2Step,
    Ipp<datatype> alpha2, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16u_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>
<code>8u_C4R</code>	<code>16u_C4R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>

```
IppStatus ippAlphaCompC_<mod>(const Ipp<datatype>* const pSrc1[4],
    int src1Step, Ipp<datatype> alpha1, const Ipp<datatype>* const pSrc2[4],
    int src2Step, Ipp<datatype> alpha2, Ipp<datatype>* const pDst[4],
    int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for *mod*:

8u\_AP4R      16u\_AP4R

## Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
<i>src1Step</i> , <i>src2Step</i>	Distances in bytes between starts of consecutive lines in the source image buffers.
<i>pDst</i>	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>alpha1</i> , <i>alpha2</i>	Constant alpha values to use for compositing operation.
<i>alphaType</i>	The composition type to perform. See <a href="#">Table 5-3</a> for the type value and description.

## Description

The function `ippiAlphaCompC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs an image compositing operation on 1-channel image buffers, 3-channel RGB and 4-channel RGBA image buffers and on planar images, using constant alpha values *alpha1* and *alpha2*. These values are passed to the function as parameters. The compositing is done by overlaying pixels from the foreground image ROI *pSrc1* with pixels from the background image ROI *pSrc2* to produce pixels in the resultant image ROI *pDst*. The alpha values are normalized to the range [0..1]. The type of the compositing operation is indicated by the *alphaType* argument. Use [Table 5-3](#) to choose a valid *alphaType* value depending on the required composition type. For example, the resulting pixel color components for the OVER operation (see [Table 5-2](#)) are computed as follows:

$$r_C = \alpha_1 * r_A + (1 - \alpha_1) * \alpha_2 * r_B$$

$$g_C = \alpha_1 * g_A + (1 - \alpha_1) * \alpha_2 * g_B$$

$$b_C = \alpha_1 * b_A + (1 - \alpha_1) * \alpha_2 * b_B$$

where  $\alpha_1, \alpha_2$  are the normalized alpha values *alpha1*, *alpha2*.

This function can be used for unsigned pixel data only.

The following code example shows how to use alpha composition function:

#### Example 5-4 Using Alpha Composition Function

---

```
IppStatus acomp( void ) {
    Ipp8u imga[8*8], imgb[8*8], imgc[8*8];
    IppiSize roi = { 8, 8 };
    IppStatus st;
    ippiImageRamp_8u_C1R( imga, 8, roi, 0, 1, ippAxsHorizontal );
    ippiImageRamp_8u_C1R( imgb, 8, roi, 0, 2, ippAxsHorizontal );
    st = ippiAlphaCompC_8u_C1R( imga, 8, 255/3, imgb, 8, 255, imgc, 8, roi,
    ippAlphaOver );
    printf( "over: a=%d,A=255/3; b=%d,B=255; c=%d //
c=a*A+b*(1-A)*B\n",imga[6],imgb[6],imgc[6] );
    return st;
}
```

#### Output

```
over: a=6,A=255/3; b=12,B=255; c=10 // c=a*A+b*B*(1-A)
```

---

#### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>roiSize</i> has a field with zero or negative value.

## AlphaPremul

*Pre-multiplies pixel values of an image by its alpha values.*

### Syntax

#### Case 1: Not-in-place operation.

```
IppStatus ippAlphaPremul_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_AC4R      16u\_AC4R

```
IppStatus ippAlphaPremul_<mod>(const Ipp<datatype>* const pSrc[4],
    int srcStep, Ipp<datatype>* const pDst[4], int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u\_AP4R      16u\_AP4R

#### Case 2: In-place operation.

```
IppStatus ippAlphaPremul_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u\_AC4IR      16u\_AC4IR

```
IppStatus ippAlphaPremul_<mod>(Ipp<datatype>* const pSrcDst[4],
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_AP4IR      16u\_AP4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.

<i>pDst</i>	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiAlphaPremul` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a RGBA source image (pixel order or planar) to the pre-multiplied alpha form. If (r,g,b,a) are the red, green, blue, and alpha values of a pixel, then new pixel values are (r\* $\alpha$ , g\* $\alpha$ , b\* $\alpha$ , a) after execution of this function. Here  $\alpha$  is the pixel's normalized alpha value in the range 0 to 1.

The function `ippiAlphaPremul` can be used for unsigned pixel data only.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>roiSize</i> has a field with zero or negative value.

## AlphaPremulC

*Pre-multiplies pixel values of an image using constant alpha (opacity) values.*

### Syntax

#### Case 1: Not-in-place operation.

```
IppStatus ippiAlphaPremulC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> alpha, Ipp<datatype>* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u_C1R	16u_C1R
8u_C3R	16u_C3R
8u_C4R	16u_C4R
8u_AC4R	16u_AC4R

```
IppStatus ippiAlphaPremulC_<mod>(const Ipp<datatype>* const pSrc[4],
    int srcStep, Ipp<datatype> alpha, Ipp<datatype>* const pDst[4],
    int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_AP4R	16u_AP4R
---------	----------

#### Case 2: In-place operation .

```
IppStatus ippiAlphaPremulC_<mod>(Ipp<datatype> alpha,
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR
8u_C3IR	16u_C3IR
8u_C4IR	16u_C4IR
8u_AC4IR	16u_AC4IR

```
IppStatus ippiAlphaPremulC_<mod>(Ipp<datatype> alpha,
    Ipp<datatype>* const pSrcDst[4], int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_AP4IR	16u_AP4IR
----------	-----------

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination image ROI for pixel-order data. An array of pointers to separate ROI in the destination color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image ROI or an array of pointers to ROI in the separate source and destination color planes for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>alpha</i>	Global alpha value used for pre-multiplying pixel values.

## Description

The function `ippiAlphaPremulC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts either a 1-, 3-, 4-channel image or planar RGBA image to the pre-multiplied alpha form, using global alpha value *alpha*.

For 1-, 3-, 4-channel image buffers, pixel values in each channel are multiplied by  $\alpha$ ; for RGBA (pixel order and planar) images with (r,g,b,a) pixel values, new pixel values are (r\* $\alpha$ , g\* $\alpha$ , b\* $\alpha$ , *alpha*) after execution of this function.

Here  $\alpha$  is the normalized *alpha* value in the range 0 to 1.

The function `ippiAlphaPremulC` can be used for unsigned pixel data only.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.



<code>ippStsSizeErr</code>	Indicates an error condition if the <i>roiSize</i> has a field with zero or negative value.
----------------------------	---

# *Image Color Conversion*

---

## 6

This chapter describes Intel IPP image processing functions that perform different type of image color conversion. The Intel IPP software supports the following image color conversions:

- Color models conversion
- Conversion from color to gray scale
- Different types of format conversion:
  - from pixel-order to planar format and vice versa
  - changing number of channels or planes
  - changing sampling format
  - altering order of samples or planes
- Gamma correction
- Reduction from high bit resolution color to low bit resolution color
- Intensity transformation using lookup tables
- Color twist

All Intel IPP color conversion functions perform point operations on pixels of the source image. For a given destination pixel, the resultant channel values are computed using channel values of the corresponding source pixel only, and not involving any neighborhood pixels. Thus, the rectangular region of interest ([ROI](#)) used in function operations may extend to the size of the whole image.

[Table 6-1](#) lists Intel IPP color space conversion functions.

Table 6-1 Color Conversion Functions

Function Base Name	Description
<b>Color Model Conversion</b>	
<a href="#"><u>RGBToYUV</u></a>	Convert RGB image to and from YUV color model.
<a href="#"><u>YUVToRGB</u></a>	
<a href="#"><u>RGBToYUV422</u></a>	Convert RGB image to and from 4:2:2 YUV image.
<a href="#"><u>YUV422ToRGB</u></a>	
<a href="#"><u>RGBToYUV420</u></a>	Convert RGB images to and from 4:2:2 YUV image.
<a href="#"><u>YUV420ToRGB</u></a>	
<a href="#"><u>YUV420ToBGR</u></a>	Convert 4:2:0 YUV image to BGR image.
<a href="#"><u>YUV420ToRGB565</u></a>	Convert 4:2:0 YUV image to the 16-bit RGB image.
<a href="#"><u>YUV420ToRGB555</u></a>	
<a href="#"><u>YUV420ToRGB444</u></a>	
<a href="#"><u>YUV420ToRGB565Dither</u></a>	Convert 4:2:0 YUV image to the 16-bit RGB image with dithering.
<a href="#"><u>YUV420ToRGB555Dither</u></a>	
<a href="#"><u>YUV420ToRGB444Dither</u></a>	
<a href="#"><u>BGR565ToYUV420</u></a>	Convert 16-bit BGR image to the 4:2:0 YUV image.
<a href="#"><u>BGR555ToYUV420</u></a>	
<a href="#"><u>YUV420ToBGR565</u></a>	Convert 4:2:0 YUV image to the 16-bit BGR image.
<a href="#"><u>YUV420ToBGR555</u></a>	
<a href="#"><u>YUV420ToBGR444</u></a>	
<a href="#"><u>YUV420ToBGR565Dither</u></a>	Convert 4:2:0 YUV image to the 16-bit BGR image with dithering.
<a href="#"><u>YUV420ToBGR555Dither</u></a>	
<a href="#"><u>YUV420ToBGR444Dither</u></a>	
<a href="#"><u>RGBToYCbCr</u></a>	Convert RGB images to and from YCbCr color model
<a href="#"><u>YCbCrToRGB</u></a>	
<a href="#"><u>YCbCrToBGR</u></a>	Converts a YCbCr image to the BGR color model.
<a href="#"><u>YCbCrToRGB565</u></a>	Convert YCbCr image to the 16-bit per pixel RGB image.
<a href="#"><u>YCbCrToRGB555</u></a>	
<a href="#"><u>YCbCrToRGB444</u></a>	
<a href="#"><u>YCbCrToRGB565Dither</u></a>	Convert YCbCr image to the 16-bit per pixel RGB image with dithering
<a href="#"><u>YCbCrToRGB555Dither</u></a>	
<a href="#"><u>YCbCrToRGB444Dither</u></a>	
<a href="#"><u>YCbCrToBGR565</u></a>	Convert YCbCr image to the 16-bit per pixel BGR image.
<a href="#"><u>YCbCrToBGR555</u></a>	
<a href="#"><u>YCbCrToBGR444</u></a>	

**Table 6-1**      **Color Conversion Functions (continued)**

Function Base Name	Description
<a href="#"><u>YCbCrToBGR565Dither</u></a> <a href="#"><u>YCbCrToBGR555Dither</u></a> <a href="#"><u>YCbCrToBGR444Dither</u></a>	Convert YCbCr image to the 16-bit per pixel BGR image with dithering.
<a href="#"><u>RGBToYCbCr422</u></a> <a href="#"><u>YCbCr422ToRGB</u></a>	Convert RGB image to and from 4:2:2 YCbCr image.
<a href="#"><u>BGRToYCbCr422</u></a> <a href="#"><u>YCbCr422ToBGR</u></a>	Converts 24-bit per pixel BGR image to 16-bit per pixel YCbCr image. Converts 16-bit per pixel YCbCr image to 24-bit per pixel BGR image.
<a href="#"><u>BGR565ToYCbCr422</u></a> <a href="#"><u>BGR555ToYCbCr422</u></a>	Converts 16-bit per pixel BGR image to 16-bit per pixel YCbCr image.
<a href="#"><u>RGBToCbYCr422</u></a> <a href="#"><u>RGBToCbYCr422Gamma</u></a> <a href="#"><u>CbYCr422ToRGB</u></a>	Convert RGB images to and from 4:2:2 CbYCr image.
<a href="#"><u>BGRToCbYCr422</u></a> <a href="#"><u>CbYCr422ToBGR</u></a>	Convert BGR images to and from CbYCr color model with 4:2:2 sampling format
<a href="#"><u>YCbCr422ToRGB565</u></a> <a href="#"><u>YCbCr422ToRGB555</u></a> <a href="#"><u>YCbCr422ToRGB444</u></a>	Convert 16-bit per pixel YCbCr image that has 4:2:2 sampling format to the 16-bit per pixel RGB image.
<a href="#"><u>YCbCr422ToRGB565Dither</u></a> <a href="#"><u>YCbCr422ToRGB555Dither</u></a> <a href="#"><u>YCbCr422ToRGB444Dither</u></a>	Convert 16-bit per pixel YCbCr image that has 4:2:2 sampling format to the 16-bit per pixel RGB image with dithering
<a href="#"><u>YCbCr422ToBGR565</u></a> <a href="#"><u>YCbCr422ToBGR555</u></a> <a href="#"><u>YCbCr422ToBGR444</u></a>	Convert 16-bit per pixel YCbCr image that has 4:2:2 sampling format to the 16-bit per pixel BGR image.
<a href="#"><u>YCbC422ToBGR565Dither</u></a> <a href="#"><u>YCbC422ToBGR555Dither</u></a> <a href="#"><u>YCbCr422ToBGR444Dither</u></a>	Convert 16-bit per pixel YCbCr image that has 4:2:2 sampling format to the 16-bit per pixel BGR image with dithering.
<a href="#"><u>RGBToYCbCr420</u></a> <a href="#"><u>YCbCr420ToRGB</u></a>	Convert RGB images to and from YCbCr color model with 4:2:0 sampling format.
<a href="#"><u>YCbCr420ToRGB565</u></a> <a href="#"><u>YCbCr420ToRGB555</u></a> <a href="#"><u>YCbCr420ToRGB444</u></a>	Convert YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel RGB image.
<a href="#"><u>YCbCr420ToRGB565Dither</u></a> <a href="#"><u>YCbCr420ToRGB555Dither</u></a> <a href="#"><u>YCbCr420ToRGB444Dither</u></a>	Convert YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel RGB image with dithering.
<a href="#"><u>BGRToYCbCr420</u></a>	Converts a BGR image to the YCbCr image with 4:2:0 sampling format.

**Table 6-1 Color Conversion Functions (continued)**

Function Base Name	Description
<a href="#"><u>YCbCr420ToBGR</u></a>	Convert YCbCr image with 4:2:0 sampling format to the RGB color model.
<a href="#"><u>BGR565ToYCbCr420,</u></a> <a href="#"><u>BGR555ToYCbCr420</u></a>	Convert a 16-bit per pixel BGR image to the YCbCr image that has 4:2:0 sampling format.
<a href="#"><u>YCbCr420ToBGR565</u></a> <a href="#"><u>YCbCr420ToBGR555</u></a> <a href="#"><u>YCbCr420ToBGR444</u></a>	Convert YCbCr image with 4:2:0 sampling format to the 16-bit per pixel BGR image.
<a href="#"><u>YCbCr420ToBGR565Dither</u></a> <a href="#"><u>YCbCr420ToBGR555Dither</u></a> <a href="#"><u>YCbCr420ToBGR444Dither</u></a>	Convert YCbCr image with 4:2:0 sampling format to the 16-bit per pixel BGR image with dithering.
<a href="#"><u>BGRToYCrCb420</u></a>	Converts a BGR image to the YCrCb image with 4:2:0 sampling format.
<a href="#"><u>BGR565ToYCrCb420</u></a> <a href="#"><u>BGR555ToYCrCb420</u></a>	Converts a 16-bit per pixel BGR image to the YCrCb image with 4:2:0 sampling format.
<a href="#"><u>BGRToYCbCr411</u></a>	Converts a BGR image to the YCbCr planar image that has a 4:1:1 sampling format.
<a href="#"><u>YCbCr411ToBGR</u></a>	Converts a YCbCr image with 4:1:1 sampling format to the RGB color model.
<a href="#"><u>BGR565ToYCbCr411,</u></a> <a href="#"><u>BGR555ToYCbCr411</u></a>	Converts a 16-bit per pixel BGR image to the YCbCr image that has 4:1:1 sampling format.
<a href="#"><u>YCbCr411ToBGR565,</u></a> <a href="#"><u>YCbCr411ToBGR555</u></a>	Convert a YCbCr image that has 4:1:1 sampling format to the 16-bit per pixel BGR image.
<a href="#"><u>RGBToXYZ</u></a> <a href="#"><u>XYZToRGB</u></a>	Convert RGB images to and from XYZ color model
<a href="#"><u>RGBToLUV</u></a> <a href="#"><u>LUVToRGB</u></a>	Convert RGB images to and from CIE LUV color model
<a href="#"><u>BGRToLab</u></a> <a href="#"><u>LabToBGR</u></a>	Convert BGR images to and from CIE Lab color model
<a href="#"><u>RGBToYCC</u></a> <a href="#"><u>YCCToRGB</u></a>	Convert RGB images to and from YCC color model
<a href="#"><u>RGBToHLS</u></a> <a href="#"><u>HLSToRGB</u></a>	Convert RGB images to and from HLS color model
<a href="#"><u>BGRToHLS</u></a> <a href="#"><u>HLSToBGR</u></a>	Converts BGR images to HLS color model Converts HLS images to RGB color model
<a href="#"><u>RGBToHSV</u></a> <a href="#"><u>HSVToRGB</u></a>	Convert RGB images to and from HSV color model
<a href="#"><u>BGRToYCoCg</u></a>	Converts a 24-bit BGR image to the YCoCg color model.

**Table 6-1**      **Color Conversion Functions (continued)**

Function Base Name	Description
<a href="#">SBGRToYCoCg</a>	Converts a 48-bit BGR image to the YCoCg color model.
<a href="#">YCoCgToBGR</a>	Converts a YCoCg image to the 24-bit BGR image.
<a href="#">YCoCgToSBGR</a>	Converts a YCoCg image to the 48-bit BGR image.
<a href="#">BGRToYCoCg_Rev</a>	Converts a 24-bit BGR image to the YCoCg-R color model.
<a href="#">SBGRToYCoCg_Rev</a>	Converts a 48-bit BGR image to the YCoCg-R color model.
<a href="#">YCoCgToBGR_Rev</a>	Converts a YCoCg-R image to the 24-bit BGR image.
<a href="#">YCoCgToSBGR_Rev</a>	Converts a YCoCg-R image to the 48-bit BGR image.
<b>Color to Gray Scale Conversion</b>	
<a href="#">RGBToGray</a>	Converts an RGB image to gray scale using fixed transform coefficients
<a href="#">ColorToGray</a>	Converts an RGB image to gray scale using custom transform coefficients
<b>Format Conversion</b>	
<a href="#">YCbCr422</a>	Converts 4:2:2 YCbCr image
<a href="#">YCbCr422ToYCrCb422</a>	Converts 4:2:2 YCbCr image to 4:2:2 YCrCb image.
<a href="#">YCbCr422ToCbYCr422</a>	Converts 4:2:2 YCbCr image to 4:2:2 CbYCr image.
<a href="#">YCbCr422ToYCbCr420</a>	Converts 4:2:2 YCbCr image to 4:2:0 YCbCr image.
<a href="#">YCbCr422ToYCrCb420</a>	Converts 4:2:2 YCbCr image to 4:2:0 YCrCb image.
<a href="#">YCbCr422ToYCbCr411</a>	Converts 4:2:2 YCbCr image to 4:1:1 YCbCr image.
<a href="#">YCrCb422ToYCbCr420</a>	Converts 4:2:2 YCrCb image to 4:2:2 YCbCr image.
<a href="#">YCrCb422ToYCbCr420</a>	Converts 4:2:2 YCrCb image to 4:2:0 YCbCr image.
<a href="#">YCrCb422ToYCbCr411</a>	Converts 4:2:2 YCrCb image to 4:1:1 YCbCr image.
<a href="#">CbYCr422ToYCbCr422</a>	Converts 4:2:2 CbYCr image to 4:2:2 YCbCr image.
<a href="#">CbYCr422ToYCbCr420</a>	Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image.
<a href="#">CbYCr422ToYCrCb420</a>	Converts 4:2:2 CbYCr image to 4:2:0 YCrCb image.
<a href="#">CbYCr422ToYCbCr411</a>	Converts 4:2:2 CbYCr image to 4:1:1 YCbCr image.
<a href="#">YCbCr420</a>	Converts 4:2:0 YCbCr image
<a href="#">YCbCr420ToYCbCr422</a>	Converts 4:2:0 YCbCr image to 4:2:2 YCbCr image.
<a href="#">YCbCr420ToYCbCr422 Filter</a>	Converts 4:2:0 YCbCr image to 4:2:2 YCbCr image with additional filtering.
<a href="#">YCbCr420ToCbYCr422</a>	Converts 4:2:0 YCbCr image to 4:2:2 CbYCr image.
<a href="#">YCbCr420ToYCrCb420</a>	Converts 4:2:0 YCbCr image to 4:2:0 YCrCb image.
<a href="#">YCbCr420ToYCrCb420 Filter</a>	Converts 4:2:0 YCbCr image to 4:2:0 YCrCb image with deinterlace filtering.

**Table 6-1 Color Conversion Functions (continued)**

Function Base Name	Description
<a href="#">YCbCr420ToYCbCr411</a>	Converts 4:2:0 YCbCr image to 4:1:1 YCbCr image.
<a href="#">YCrCb420ToYCbCr422</a>	Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image.
<a href="#">YCrCb420ToYCbCr422_Filter</a>	Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image with additional filtering.
<a href="#">YCrCb420ToCbYCr422</a>	Converts 4:2:0 YCrCb image to 4:2:2 CbYCr image.
<a href="#">YCrCb420ToYCbCr420</a>	Converts 4:2:0 YCrCb image to 4:2:0 YCbCr image.
<a href="#">YCrCb420ToYCbCr411</a>	Converts 4:2:0 YCrCb image to 4:1:1 YCbCr image.
<a href="#">YCbCr411</a>	Converts 4:1:1 YCbCr image
<a href="#">YCbCr411ToYCbCr422</a>	Converts 4:1:1 YCbCr image to 4:2:2 YCbCr image.
<a href="#">YCbCr411ToYCrCb422</a>	Converts 4:1:1 YCbCr image to 4:2:2 YCrCb image.
<a href="#">YCbCr411ToYCbCr420</a>	Converts 4:1:1 YCbCr image to 4:2:0 YCbCr image.
<a href="#">YCbCr411ToYCrCb420</a>	Converts 4:1:1 YCbCr image to 4:2:0 YCrCb image.
<b>Gamma Correction</b>	
<a href="#">GammaFwd</a>	Performs gamma-correction of the source RGB image
<a href="#">GammaInv</a>	Converts a gamma-corrected R'G'B' image back to the original RGB image
<b>Reducing Bit Resolution</b>	
<a href="#">ReduceBits</a>	Reduces the bit resolution of an image
<b>Lookup Table Conversion</b>	
<a href="#">LUT</a>	Maps an image by applying intensity transformation
<a href="#">LUT_Linear</a>	Maps an image by applying intensity transformation with linear interpolation
<a href="#">LUT_Cubic</a>	Maps an image by applying intensity transformation with cubic interpolation
<a href="#">LUTPalette</a>	Maps an image by applying intensity transformation in accordance with a palette table.
<b>Color Twist</b>	
<a href="#">ColorTwist</a>	Applies a color-twist matrix to an image with integer pixel values
<a href="#">ColorTwist32f</a>	Applies a color-twist matrix to an image with floating-point pixel values

This chapter starts with introductory material that discusses color space models essential for understanding the Intel IPP color conversion functions.

For more information about color spaces and color conversion techniques, see [[Jack01](#)], [[Rogers85](#)], and [[Foley90](#)].

## Gamma Correction

The luminance intensity generated by most displays is not a linear function of the applied signal but is proportional to some power (referred to as *gamma*) of the signal voltage. As a result, high intensity ranges are expanded and low intensity ranges are compressed. This nonlinearity must be compensated for in order to achieve correct color reproduction. To do this, luminance of each of the linear red, green, and blue components is reduced to a nonlinear form using an inverse transformation. This process is called "gamma correction".

The Intel IPP functions use the following basic equations to convert an RGB image to the gamma-corrected R'G'B' image:

for  $R, G, B < 0.018$

$$R' = 4.5R$$

$$G' = 4.5G$$

$$B' = 4.5B$$

for  $R, G, B \geq 0.018$

$$R' = 1.099R^{0.45} - 0.099$$

$$G' = 1.099G^{0.45} - 0.099$$

$$B' = 1.099B^{0.45} - 0.099$$

Note that the channel intensity values are normalized to fit in the range [0..1]. The gamma value is equal to  $1/0.45 = 2.22$  in conformity with ITU Rec.709 specification (see [[ITU709](http://www.itu.int/rec/T-REC-709)]).

## CIE Chromaticity Diagram and Color Gamut

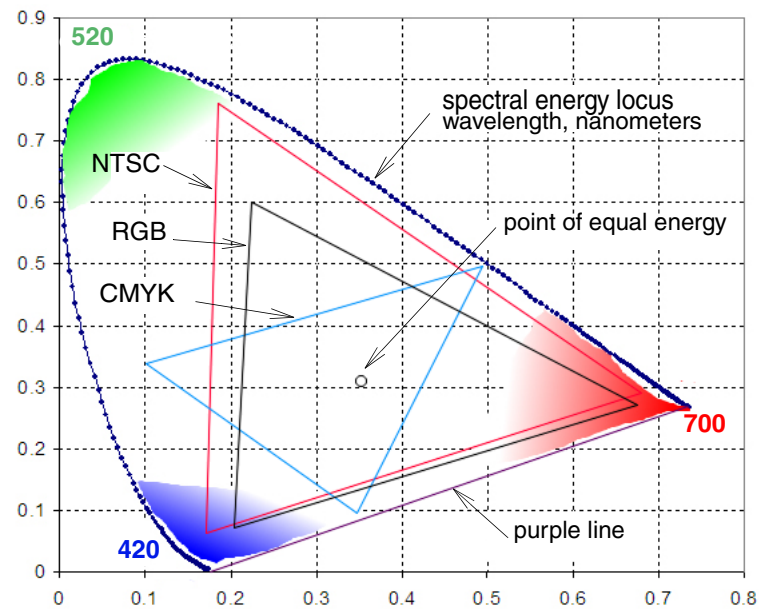
[Figure 6-1](#) presents a diagram of all visible colors. It is called a chromaticity diagram and was developed as a result of the experimental investigations performed by CIE (International Commission on Illumination, <http://members.eunet.at/cie>). The diagram presents visible colors as a function of  $x$  (red) and  $y$  (green) components called *chromaticity coordinates*. Positions of various spectrum colors (from violet to red) are indicated as the points of tongue-shaped curve called *spectrum locus*. The straight line connecting the ends of the curve is called the *purple line*. The point of equal energy represents the CIE standard for white light. Any point within the diagram represents some mixture of spectrum colors. The pure or fully saturated colors lie on the spectrum locus. Straight-line segment joining any two points in the diagram defines all color variations than can be obtained by additively combining these two colors. A triangle with vertices at any three points determine the gamut of colors that can be obtained by combining corresponding three colors.



The structure of the human eye that distinguishes three different stimuli, establishes the three-dimensional nature of color. The color may be described with a set of three parameters called tristimulus values, or components. These values may, for example, be dominant wavelength, purity, and luminance, or so-called primary colors: red, green, and blue.

The chromaticity diagram exhibits that the gamut of any three fixed colors can not enclose all visible colors. For example, [Figure 6-1](#) shows schematically the gamut of reproducible colors for the RGB primaries of a typical color CRT monitor, CMYK color printing, and for the NTSC television.

**Figure 6-1 CIE  $xyY$  Chromaticity Diagram and Color Gamut**



## Color Models

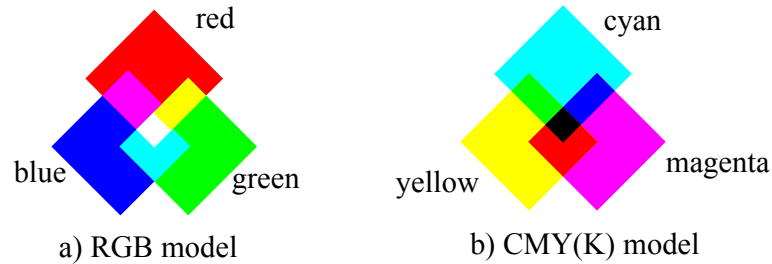
The purpose of a color model is to facilitate the specification of colors in some standard generally accepted way. In essence, a color model is a specification of a 3-D coordinate system and a subspace within that system where each color is represented by a single point.

Each industry that uses color employs the most suitable color model. For example, RGB color model is used in computer graphics, and YUV or YCbCr are used in video systems, PhotoYCC\* is used in PhotoCD\* production and so on. Transferring color information from one industry to another requires transformation from one set of values to another. The Intel IPP provides a wide number of functions to convert different color spaces to RGB and vice versa.

### RGB Color Model

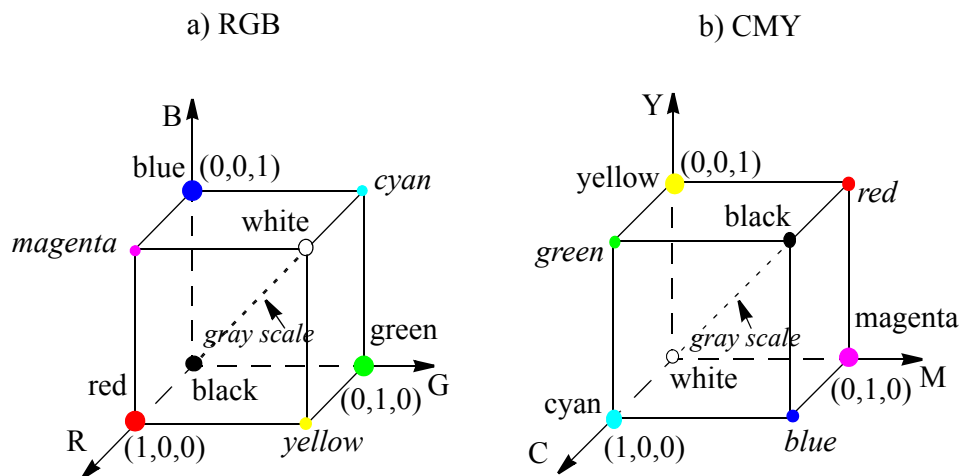
In the RGB model, each color appears as a combination of red, green, and blue. This model is called additive, and the colors are called primary colors. The primary colors can be added to produce the secondary colors of light (see [Figure 6-2](#)) - magenta (red plus blue), cyan (green plus blue), and yellow (red plus green). The combination of red, green, and blue at full intensities makes white. The color subspace of interest is a cube shown in [Figure 6-3](#) (RGB values are normalized to 0..1), in which RGB values are at three corners; cyan, magenta, and yellow are the three other corners, black is at their origin; and white is at the corner farthest from the origin.

**Figure 6-2 Primary and Secondary Colors for RGB and CMYK Models**



The gray scale extends from black to white along the diagonal joining these two points. The colors are the points on or inside the cube, defined by vectors extending from the origin.

**Figure 6-3 RGB and CMY Color Models**



Thus, images in the RGB color model consist of three independent image planes, one for each primary color.

As a rule, Intel IPP color conversion functions operate with non-linear [gamma-corrected](#) images  $R'G'B'$ .

The importance of the RGB color model is that it relates very closely to the way that the human eye perceives color. RGB is a basic color model for computer graphics because color displays use red, green, and blue to create the desired color. Therefore, the choice of the RGB color space simplifies the architecture and design of the system. Also, a system that is designed using the RGB color space can take advantage of a large number of existing software routines, since this color space has been around for a number of years.

However, RGB is not very efficient when dealing with real-world images. To generate any color within the RGB color cube, all three RGB components need to be of equal pixel depth and display resolution. Also, any modification of the image requires modification of all three planes.

## CMYK Color Model

The CMYK color model is a subset of the RGB model and is primarily used in color print production. CMYK is an acronym for cyan, magenta, and yellow along with black (noted as K). The CMYK color space is subtractive, meaning that cyan, magenta yellow, and black pigments or inks are applied to a white surface to subtract some color from white surface to create the final color. For example (see [Figure 6-2](#)), cyan is white minus red, magenta is white minus green, and yellow is white minus blue. Subtracting all colors by combining the CMY at full saturation should, in theory, render black. However, impurities in the existing CMY inks make full and equal saturation impossible, and some RGB light does filter through, rendering a muddy brown color. Hence, the addition of black ink to CMY. The CMY cube is shown in [Figure 6-3](#), in which CMY values are at three corners; red, green, and blue are the three other corners, white is at the origin; and black is at the corner farthest from the origin.

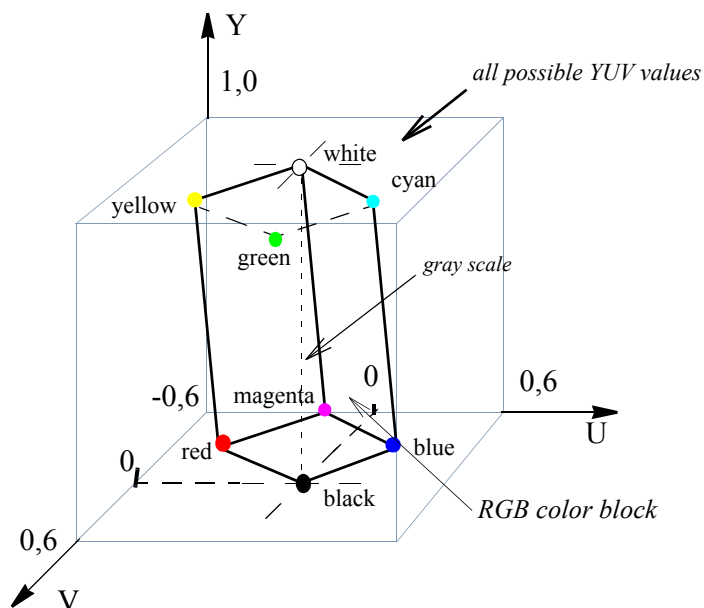
## YUV Color Model

The YUV color model is the basic color model used in analogue color TV broadcasting. Initially YUV is the re-coding of RGB for transmission efficiency (minimizing bandwidth) and for downward compatibility with black-and white television. The YUV color space is “derived” from the RGB space. It comprises the *luminance* (Y) and two color difference (U, V) components. The luminance can be computed as a weighted sum of red, green and blue components; the color difference, or *chrominance*, components are formed by subtracting luminance from blue and from red.

The principal advantage of the YUV model in image processing is decoupling of luminance and color information. The importance of this decoupling is that the luminance component of an image can be processed without affecting its color component. For example, the histogram equalization of the color image in YUV format may be performed simply by applying histogram equalization to its Y component.

There are many combinations of YUV values from nominal ranges that result in invalid RGB values, since the possible RGB colors occupy only part of the YUV space limited by these ranges. [Figure 6-4](#) shows the valid color block in the YUV space that corresponds to the RGB color cube RGB values are normalized to [0..1]).

Figure 6-4 RGB Colors Cube in the YUV Color Space



The Y'U'V' notation means that the components are derived from gamma-corrected R'G'B'. Weighted sum of these non-linear components forms a signal representative of luminance that is called *luma* Y'. (*Luma* is often loosely referred to as *luminance*, so you need be careful to determine whether a particular author assigns a linear or non-linear interpretation to the term *luminance*).

The Intel IPP functions use the following basic equation [Jack01] to convert between gamma-corrected R'G'B' and Y'U'V' models:

$$\begin{aligned} Y' &= 0.299 \cdot R' + 0.587 \cdot G' + 0.114 \cdot B' \\ U' &= -0.147 \cdot R' - 0.289 \cdot G' + 0.436 \cdot B' = 0.492 \cdot (B' - Y') \\ V' &= 0.615 \cdot R' - 0.515 \cdot G' - 0.100 \cdot B' = 0.877 \cdot (R' - Y') \\ R' &= Y' + 1.140 \cdot V' \\ G' &= Y' - 0.394 \cdot U' - 0.581 \cdot V' \\ B' &= Y' + 2.032 \cdot U' \end{aligned}$$

There are several YUV sampling formats such as 4:4:4, 4:2:2, and 4:2:0 that are supported by the Intel IPP color conversion functions and are described later in this chapter (see [“Image Downsampling”](#)).

## YCbCr and YCCK Color Models

The YCbCr color space is used for component digital video and was developed as part of the ITU-R BT.601 Recommendation. YCbCr is a scaled and offset version of the YUV color space.

The Intel IPP functions use the following basic equations [Jack01] to convert between  $R'G'B'$  in the range 0-255 and  $Y'Cb'Cr'$  (this notation means that all components are derived from gamma-corrected  $R'G'B'$ ):

$$\begin{aligned} Y' &= 0.257 * R' + 0.504 * G' + 0.098 * B' + 16 \\ Cb' &= -0.148 * R' - 0.291 * G' + 0.439 * B' + 128 \\ Cr' &= 0.439 * R' - 0.368 * G' - 0.071 * B' + 128 \\ R' &= 1.164 * (Y' - 16) + 1.596 * (Cr' - 128) \\ G' &= 1.164 * (Y' - 16) - 0.813 * (Cr' - 128) - 0.392 * (Cb' - 128) \\ B' &= 1.164 * (Y' - 16) + 2.017 * (Cb' - 128) \end{aligned}$$

Intel IPP color conversion functions specific for JPEG codec used different equations:

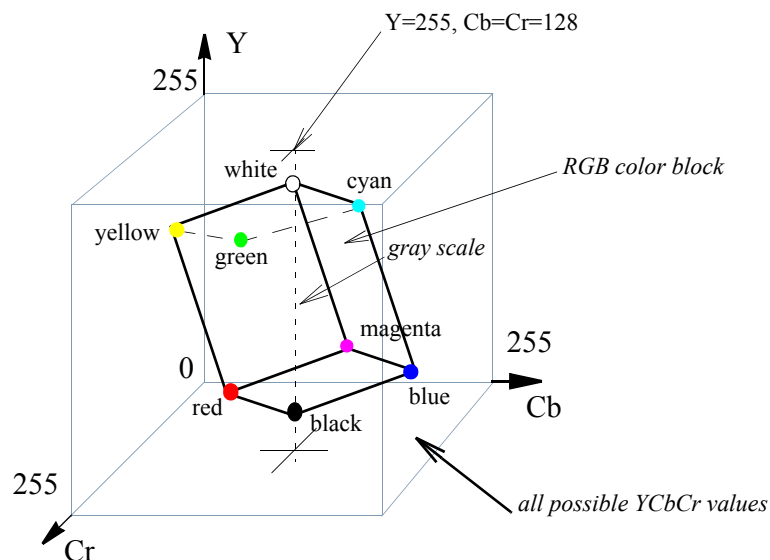
$$\begin{aligned} Y &= 0.299 * R + 0.5587 * G + 0.114 * B \\ Cb &= -0.116874 * R - 0.33126 * G + 0.5 * B + 128 \\ Cr &= 0.5 * R - 0.41869 * G - 0.08131 * B + 128 \end{aligned}$$

$$\begin{aligned} R &= Y + 1.402 * Cr - 179.456 \\ G &= Y - 0.34414 * Cb - 0.71414 * Cr + 135.45984 \\ B &= Y + 1.772 * Cb - 226.816 \end{aligned}$$

YCCK model is specific for the JPEG image compression. It is a variant of YCbCr model containing an additional  $K$  channel (black). The fact is that JPEG codec performs more effectively if the luminance and color information are decoupled. Therefore a CMYK image should be converted to YCCK before JPEG compression (see description of the [CMYKToYCCK JPEG](#) function in Chapter 15 for more details).

The possible RGB colors occupy only part of the YCbCr color space (see [Figure 6-5](#)) limited by the nominal ranges, therefore there are many YCbCr combinations that result in the invalid RGB values.

**Figure 6-5** RGB Colors Cube in the YCbCr Space



There are several YCbCr sampling formats such as 4:4:4, 4:2:2, 4:1:1, and 4:2:0, which are supported by the Intel IPP color conversion functions and are described later in this chapter (see [“Image Downsampling”](#)).

### PhotoYCC Color Model

The Kodak\* PhotoYCC\* was developed for encoding Photo CD\* image data. It is based on both the ITU Recommendations 601 and 709, using luminance-chrominance representation of color like in BT.601 YCbCr and BT.709 ([[ITU709](#)]). This model comprises luminance (Y) and two color difference, or chrominance (C1, C2) components. The PhotoYCC is optimized for the color photographic material, and provides a color gamut that is greater than that which can currently be displayed.

The Intel IPP functions use the following basic equations [[Jack01](#)] to convert non-linear gamma-corrected  $R'G'B'$  to  $Y'C'C'$ :

$$\begin{aligned} Y' &= 0.213 * R' + 0.419 * G' + 0.081 * B' \\ C1' &= -0.131 * R' - 0.256 * G' + 0.387 * B' + 0.612 \\ C2' &= 0.373 * R' - 0.312 * G' - 0.061 * B' + 0.537 \end{aligned}$$

The equations above are given on the assumption that  $R'$ ,  $G'$ , and  $B'$  values are normalized to the range  $[0..1]$ .

Since PhotoYCC model attempts to preserve the dynamic range of film, decoding PhotoYCC images requires the selection of a color space and range appropriate for the output device. Thus, the decoding equations are not always the exact inverse of the encoding equations. The following equations [[Jack01](#)] are used in Intel IPP to generate  $R'G'B'$  values for driving a CRT display and assume a unity relationship between the luma in the encoded image and the displayed image:

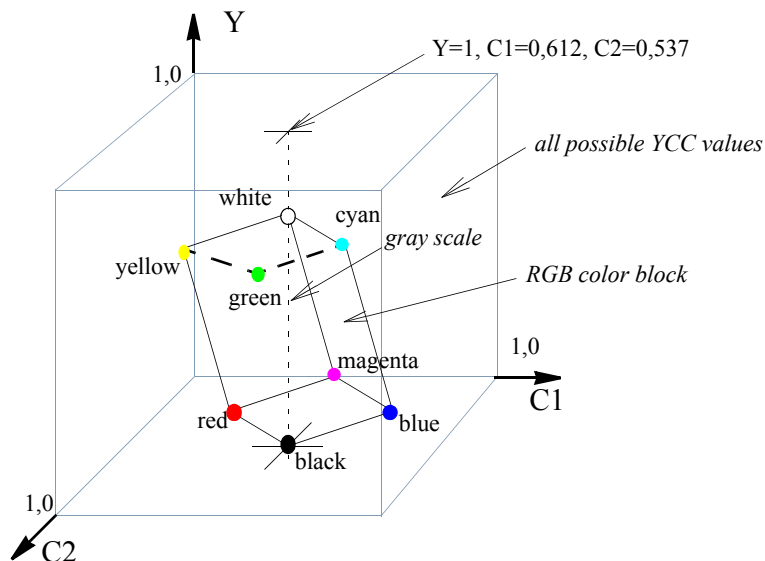
$$\begin{aligned} R' &= 0.981 * Y + 1.315 * (C2 - 0.537) \\ G' &= 0.981 * Y - 0.311 * (C1 - 0.612) - 0.669 * (C2 - 0.537) \\ B' &= 0.981 * Y + 1.601 * (C1 - 0.612) \end{aligned}$$

The equations above are given on the assumption that source  $Y$ ,  $C1$ , and  $C2$  values are normalized to the range  $[0..1]$ , and the display primaries have the chromaticity values in accordance with [[ITU709](#)] specifications.

The possible RGB colors occupy only part of the YCC color space (see [Figure 6-6](#)) limited by the nominal ranges, hence there are many YCbCr combinations that result in the invalid RGB values



**Figure 6-6 RGB Colors in the YCC Color Space**



## YCoCg Color Models

The YCoCg color model was developed to increase the effectiveness of the image compression [Malvar03]. This color model comprises the luminance (Y) and two color difference components (Co - offset orange, Cg - offset green).

The Intel IPP functions use the following simple basic equations [Malvar03] to convert between RGB and YCoCg:

$$Y = R/4 + G/2 + B/4$$

$$Co = R/2 - B/2$$

$$Cg = -R/4 + G/2 - B/4$$

$$R = Y + Co - Cg$$

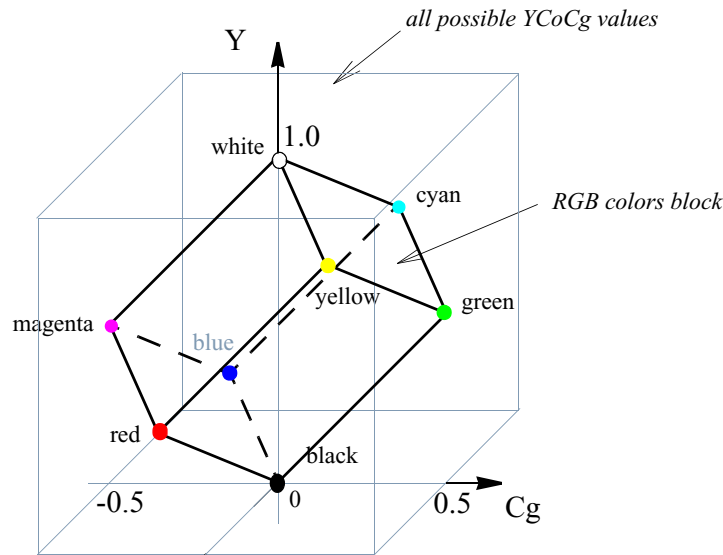
$$G = Y + Cg$$

$$B = Y - Co - Cg$$

A variation of this color space which is called YCoCg -R, enables transformation reversibility with a smaller dynamic range requirements than does YCoCg [Malvar03-1].

The possible RGB colors occupy only part of the YCoCg color space (see [Figure 6-7](#)) limited by the nominal ranges, hence there are many YCoCg combinations that result in the invalid RGB values.

**Figure 6-7 RGB Color Cube in the YCoCg Color Space**



## HSV, and HLS Color Models

The HLS (hue, lightness, saturation) and HSV (hue, saturation, value) color models were developed to be more “intuitive” in manipulating with color and were designed to approximate the way humans perceive and interpret color.

*Hue* defines the color itself. The values for the hue axis vary from 0 to 360 beginning and ending with red and running through green, blue and all intermediary colors.

*Saturation* indicates the degree to which the hue differs from a neutral gray. The values run from 0, which means no color saturation, to 1, which is the fullest saturation of a given hue at a given illumination.

Intensity component - *lightness* (HLS) or *value* (HSV), indicates the illumination level. Both vary from 0 (black, no light) to 1 (white, full illumination). The difference between the two is that maximum saturation of hue is at  $S=1$  and full illumination ( $V=1$ ) in the HSV color model, whereas in the HLS color model maximum saturation is at lightness  $L=0.5$ .

The HSV color space is essentially a cylinder, but usually it is represented as a cone or hexagonal cone (hexcone) as shown in the [Figure 6-8](#), because the hexcone defines the subset of the HSV space with valid RGB values. The *value* V is the vertical axis, and the vertex V=0 corresponds to black color. Similarly, a color solid, or 3D-representation, of the HLS model is a double hexcone ([Figure 6-9](#)) with *lightness* as the axis, and the vertex of the second hexcone corresponding to white.

**Figure 6-8** HSV Solid

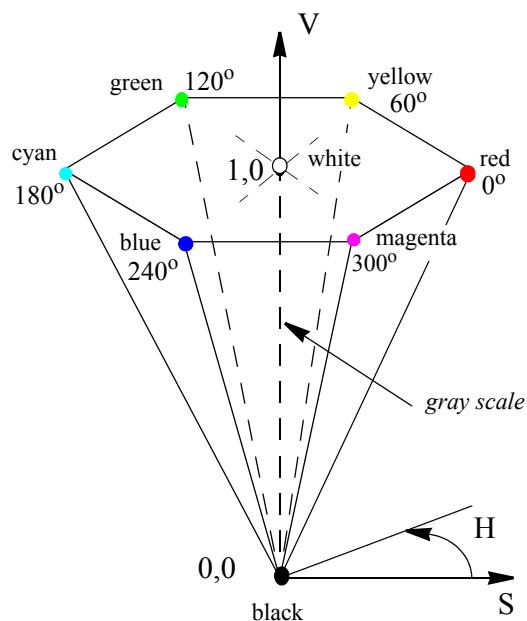
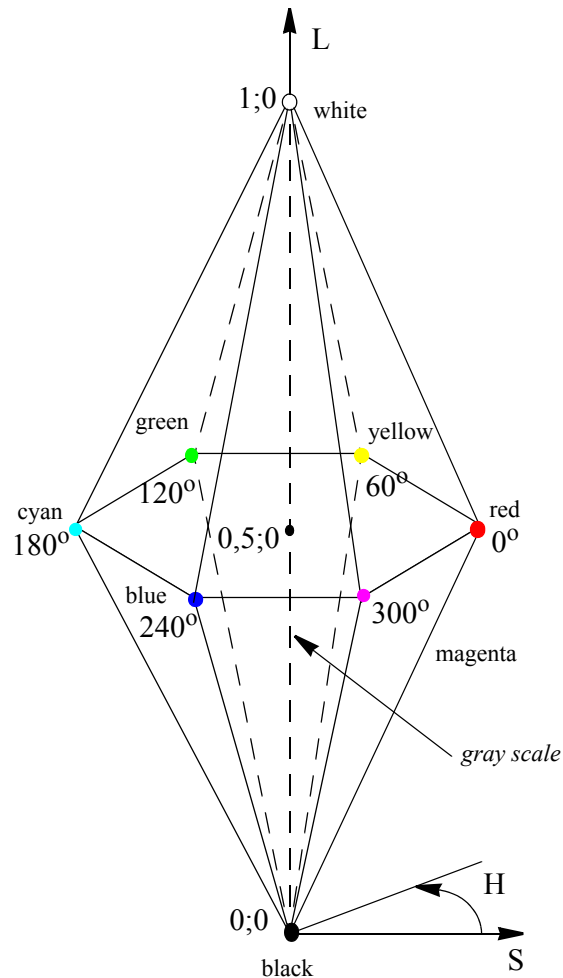


Figure 6-9 HLS Solid



Both color models have intensity component decoupled from the color information. The HSV color space yields a greater dynamic range of saturation.

Conversions from [RGB to HSV/HLS](#) and vice-versa in Intel IPP are performed in accordance with the respective pseudocode algorithms [[Rogers85](#)], given in the descriptions of corresponding conversion functions.

## CIE XYZ Color Model

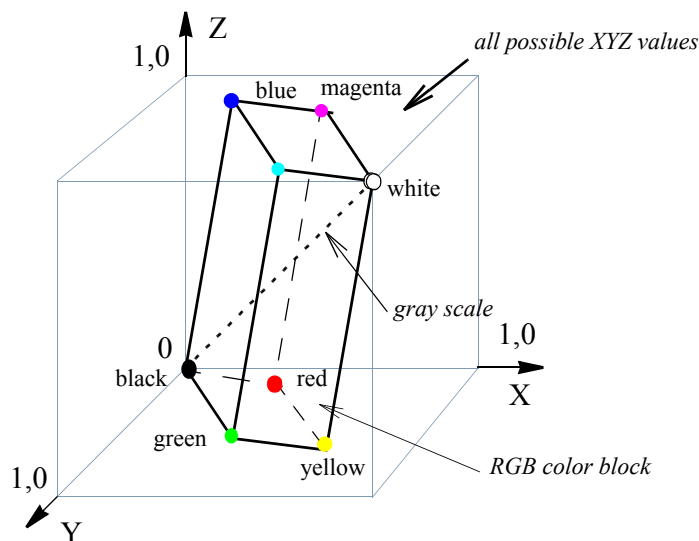
The XYZ color space is an international standard developed by the CIE (Commission Internationale de l'Eclairage). This model is based on three hypothetical primaries, XYZ, and all visible colors can be represented by using only positive values of X, Y, and Z. The CIE XYZ primaries are hypothetical because they do not correspond to any real light wavelengths. The Y primary is intentionally defined to match closely to luminance, while X and Z primaries give color information.

The main advantage of CIE XYZ space (and any color space based on it) is that this space is completely device independent. The chromaticity diagram on [Figure 6-1](#) is in fact a two-dimensional projection of the CIE XYZ sub-space.

Note that arbitrarily combining X, Y, and Z values within nominal ranges can easily lead to a "color" outside of the visible color spectrum.

The position of the block of RGB-representable colors in the XYZ space is shown in the [Figure 6-10](#).

**Figure 6-10 RGB Colors Cube in the XYZ Color Space**



Intel IPP functions use the following basic equations [[Rogers85](#)], to convert between gamma-corrected R'G'B' and CIE XYZ models:

$$\begin{aligned} X &= 0.412453 * R' + 0.35758 * G' + 0.180423 * B' \\ Y &= 0.212671 * R' + 0.71516 * G' + 0.072169 * B' \\ Z &= 0.019334 * R' + 0.119193 * G' + 0.950227 * B' \end{aligned}$$

The equations for  $X, Y, Z$  calculation are given on the assumption that  $R', G',$  and  $B'$  values are normalized to the range  $[0..1]$ .

$$\begin{aligned} R' &= 3.240479 * X - 1.53715 * Y - 0.498535 * Z \\ G' &= -0.969256 * X + 1.875991 * Y + 0.041556 * Z \\ B' &= 0.055648 * X - 0.204043 * Y + 1.057311 * Z \end{aligned}$$

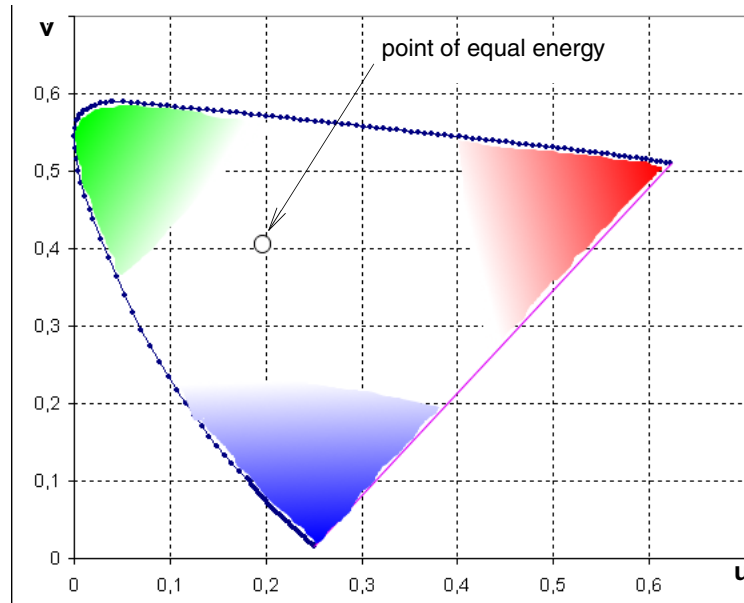
The equations for  $R', G', B'$  calculation are given on the assumption that  $X, Y,$  and  $Z$  values are in the range  $[0..1]$ .

## CIE LUV and CIE Lab Color Models

The CIE LUV and CIE Lab color models are considered to be perceptually uniform and are referred to as uniform color models. Both are uniform derivations from the standard CIE XYZ space. “Perceptually uniform” means that two colors that are equally distant in the color space are equally distant perceptually. To accomplish this approach, a uniform chromaticity scale (UCS) diagram was proposed by CIE ([Figure 6-11](#)). The UCS diagram uses a mathematical formula to transform the XYZ values or  $x, y$  coordinates ([Figure 6-1](#)), to a new set of values that present a visually more accurate two-dimensional model.

The  $Y$  lightness scale is replaced with a new scale called  $L$  that is approximately uniformly spaced but is more indicative of the actual visual differences. Chrominance components are  $U$  and  $V$  for CIE LUV, and  $a$  and  $b$  (referred also respectively as red/blue and yellow/blue chrominancies) in CIE Lab. Both color spaces are derived from the CIE XYZ color space.

Figure 6-11 CIE  $u',v'$  Uniform Chromaticity Scale Diagram



The CIE LUV color space is derived from CIE XYZ as follows ([Rogers85]),

$$L = 116. \cdot (Y/Y_n)^{1/3} - 16.$$

$$U = 13. \cdot L \cdot (u - u_n)$$

$$V = 13. \cdot L \cdot (v - v_n)$$

where

$$u = 4. \cdot X / (X + 15. \cdot Y + 3. \cdot Z)$$

$$v = 9. \cdot Y / (X + 15. \cdot Y + 3. \cdot Z)$$

$$u_n = 4. \cdot x_n / (-2. \cdot x_n + 12. \cdot y_n + 3. \cdot z_n)$$

$$v_n = 9. \cdot y_n / (-2. \cdot x_n + 12. \cdot y_n + 3. \cdot z_n)$$

Inverse conversion is performed in accordance with equations:

$$Y = Y_n \cdot ((L + 16.) / 116.)^3.$$

$$X = -9. \cdot Y \cdot u / ((u - 4.) \cdot v - u \cdot v)$$

$$Z = (9. \cdot Y - 15 \cdot v \cdot Y - v \cdot X) / 3. \cdot v$$

where

$$u = U / (13. * L) + u_n$$

$$v = V / (13. * L) + v_n$$

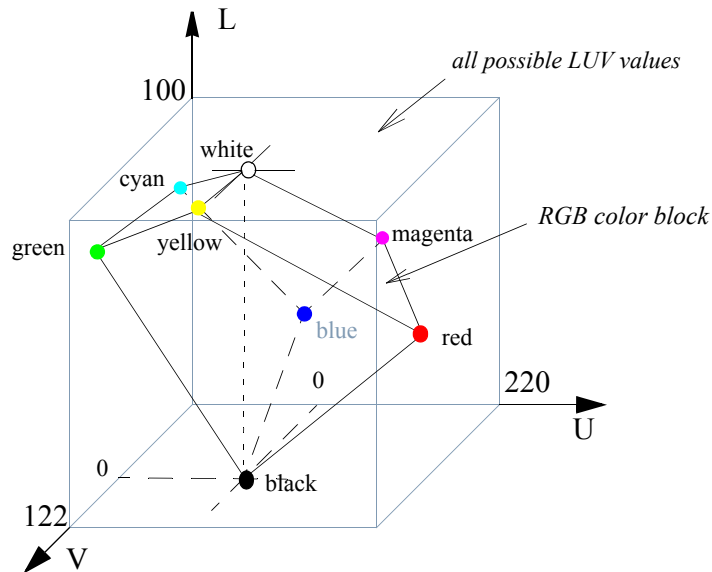
and  $u_n, v_n$  are defined above.

Here  $x_n = 0.312713$ ,  $y_n = 0.329016$  are the CIE chromaticity coordinates of the D65 white point ([ITU709]), and  $y_n = 1.0$  is the luminance of the D65 white point.

The values of the L component are in the range [0..100], U component in the range [-134..220], and V component in the range [-140..122].

The RGB-representable colors occupy only part of the LUV color space (see [Figure 6-12](#)) limited by the nominal ranges, therefore there are many LUV combinations that result in the invalid RGB value.

**Figure 6-12 RGB Color Cube in the CIE LUV Color Space**





The CIE Lab color space is derived from CIE XYZ as follows:

$$\begin{aligned} L &= 116. \cdot (Y/Y_n)^{1/3} \cdot - 16 && \text{for } Y/Y_n > 0.008856 \\ L &= 903.3 \cdot (Y/Y_n)^{1/3} && \text{for } Y/Y_n \leq 0.008856 \end{aligned}$$

$$a = 500. \cdot [f(X/X_n) - f(Y/Y_n)]$$

$$b = 200. \cdot [f(Y/Y_n) - f(Z/Z_n)]$$

where

$$\begin{aligned} f(t) &= t^{1/3} \cdot - 16 && \text{for } t > 0.008856 \\ f(t) &= 7.787 \cdot t + 16/116 && \text{for } t \leq 0.008856 \end{aligned}$$

Here  $Y_n = 1.0$  is the luminance, and  $X_n = 0.950455$ ,  $Z_n = 1.088753$  are the chrominances for the D65 white point.

The values of the L component are in the range [0..100], a and b component values are in the range [-128..127].

Inverse conversion is performed in accordance with equations:

$$\begin{aligned} Y &= Y_n \cdot P^3 \cdot \\ X &= X_n \cdot (P + a/500.)^3 \cdot \\ Z &= Z_n \cdot (P - b/200.)^3 \cdot \end{aligned}$$

where

$$P = (L + 16) / 116.$$

## Image Downsampling

Conventionally, digital color images are represented by setting specific values of the color space coordinates for each pixel. Color spaces with decoupled luminance and chrominance coordinates (YUV type) allow to reduce the number of bits required for acceptable color description of an image. This reduction is based on greater sensitivity of the human eye to changes in luminance than to changes in chrominance.

The idea behind this approach is to set individual value of luminance component to each pixel, while assigning the same color (chrominance components) to certain groups of pixels (sometimes called *macropixels*) in accordance with some specific rules. This process is called *downsampling*, and there are different sampling formats depending on the underlying scheme.

The following sampling formats are supported by Intel IPP image processing functions (excluding JPEG functions):

**4:4:4 YUV (YCbCr)** - conventional format, no downsampling, Y, U(Cb), V(Cr) components are sampled at every pixel. If each component takes 8 bits, then every pixel requires 24 bits. This format is often denoted as YUV (YCbCr) with “4:4:4” descriptor omitted.

**4:2:2 YUV (YCbCr)** - uses 2:1 horizontal downsampling. It means that the Y component is sampled at each pixel, while U(Cb) and V(Cr) components are sampled every 2 pixels in horizontal direction. If each component takes 8 bits, the pair of pixels requires 32 bits.

**4:1:1 YCbCr** - uses 4:1 horizontal downsampling. It means that the Y component is sampled at each pixel, while Cb and Cr components are sampled every 4 pixels horizontally. If each component takes 8 bits, each four horizontal pixels require 48 bits.

**4:2:0 YUV (YCbCr)** - uses 2:1 horizontal downsampling and 2:1 vertical downsampling. Y is sampled at each pixel, U(Cb) and V(Cr) are sampled at every block of 2x2 pixels. If each component takes 8 bits, each four-pixel block requires 48 bits.

In JPEG compression, downsampling has specific distinctive features and is denoted in a slightly different way. In JPEG domain, sampling formats determine the structure of minimal coded units, or MCUs. Hence, Intel IPP functions specific for JPEG codec, support the following sampling formats:

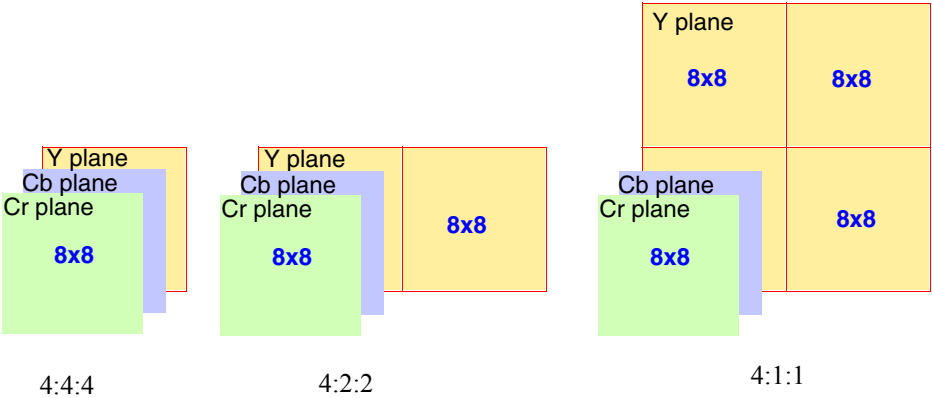
**4:4:4 YCbCr** - for every 8x8 block of Y samples, there is one 8x8 block of each Cb and Cr samples.

**4:2:2 YCbCr** - for every two horizontal 8x8 blocks of Y samples, there is one 8x8 block of each Cb and Cr samples.

**4:1:1 YCbCr** - for every four (two in horizontal and two in vertical direction) 8x8 blocks of Y samples, there is one 8x8 block of each Cb and Cr samples.

Structure of the corresponding MCU for each of these sampling formats is shown in [Figure 6-13](#).

**Figure 6-13** MCU Structure for Different JPEG Sampling Formats



### RGB Image Formats

In addition to the 24-bit-per-pixel RGB/BGR image formats, Intel IPP color conversion functions support 32-bit-per-pixel RGB/BGR formats (which include three RGB channels plus alpha channel). For 24-bit formats, each color is one byte, every pixel is three bytes. For 32-bit formats, each color is one byte and alpha component is one byte, which yields four bytes per pixel. Memory layout for these formats is given in [Table 6-2](#).

For 16-bit formats, every pixel is two bytes and each color occupies a specified number of bits. [Figure 6-14](#) shows all supported 16-bit-per-pixel formats and their memory layout (bit order):

Figure 6-14     16-bit pixel formats

16-bit pixel formats	
	High-order byte                      Low-order byte
RGB565	B4 B3 B2 B1 B0 G5 G4 G3      G2 G1 G0 R4 R3 R2 R1 R0
RGB555	X B4 B3 B2 B1 B0 G4 G3      G2 G1 G0 R4 R3 R2 R1 R0
RGB444	X X X X B3 B2 B1 B0      G3 G2 G1 G0 R3 R2 R1 R0
BGR565	R4 R3 R2 R1 R0 G5 G4 G3      G2 G1 G0 B4 B3 B2 B1 B0
BGR555	X R4 R3 R2 R1 R0 G4 G3      G2 G1 G0 B4 B3 B2 B1 B0
BGR444	X X X X R3 R2 R1 R0      G3 G2 G1 G0 B3 B2 B1 B0
R - Red, G - Green, B - Blue, X - free bit	

Pixel and Planar Image Formats

Data storage for an image can be pixel-oriented or planar-oriented (planar). For images in pixel order, all channel values for each pixel are clustered and stored consecutively. Their layout depends on the color model and downsampling scheme.

[Table 6-2](#) lists all pixel-order image formats that are supported by Intel IPP color conversion functions and shows the corresponding channel values order (here, group of underlined symbols represents one pixel and symbol *A* denotes alpha-channel value). Last column of this table gives an example of Intel IPP color conversion function that uses the respective image format.

**Table 6-2 Pixel-Order Image Formats**

Image Format	Number of Channels	Channel Values Order						Example
RGB	3	<u>R0</u>	<u>G0</u>	<u>B0</u>	<u>R1</u>	<u>G1</u>	<u>B1</u>	ippiRGBToYUV_8u_C3R
RGB444								ippiYCbCrToRGB444_8u16u_C3R
RGB555								ippiYCbCrToRGB555_8u16u_C3R
RGB565								ippiYCbCrToRGB565_8u16u_C3R
RGB	4	<u>R0</u>	<u>G0</u>	<u>B0</u>	<u>A0</u>	<u>R1</u>	<u>G1</u>	ippiRGBToYUV_8u_AC4R
BGR	3	<u>B0</u>	<u>G0</u>	<u>R0</u>	<u>B1</u>	<u>G1</u>	<u>R1</u>	ippiYCbCrToBGR_8u_P3C3R
BGR444								ippiYCbCrToBGR444_8u16u_C3R
BGR555								ippiYCbCrToBGR555_8u16u_C3R
BGR565								ippiYCbCrToBGR565_8u16u_C3R
BGR	4	<u>B0</u>	<u>G0</u>	<u>R0</u>	<u>A0</u>	<u>B1</u>	<u>G1</u>	ippiBGRToHLS_8u_AC4R
YUV	3	<u>Y0</u>	<u>U0</u>	<u>V0</u>	<u>Y1</u>	<u>U1</u>	<u>V1</u>	ippiYUVToRGB_8u_C3R
YUV	4	<u>Y0</u>	<u>U0</u>	<u>V0</u>	<u>A0</u>	<u>Y1</u>	<u>U1</u>	ippiYUVToRGB_8u_AC4R
4:2:2 YUV	2	<u>Y0</u>	<u>U0</u>	<u>Y1</u>	<u>V0</u>	<u>Y2</u>	<u>U1</u>	ippiYUV422ToRGB_8u_C2C3R
YCbCr	3	<u>Y0</u>	<u>Cb0</u>	<u>Cr0</u>	<u>Y1</u>	<u>Cb1</u>	<u>Cr1</u>	ippiYCbCrToRGB_8u_C3R
YCbCr	4	<u>Y0</u>	<u>Cb0</u>	<u>Cr0</u>	<u>A0</u>	<u>Y1</u>	<u>Cb1</u>	ippiYCbCrToRGB_8u_AC4R
4:2:2 YCbCr	2	<u>Y0</u>	<u>Cb0</u>	<u>Y1</u>	<u>Cr0</u>	<u>Y2</u>	<u>Cb1</u>	ippiYCbCr422ToRGB_8u_C2C3R
4:2:2 YCrCb	2	<u>Y0</u>	<u>Cr0</u>	<u>Y1</u>	<u>Cb0</u>	<u>Y2</u>	<u>Cr1</u>	ippiYCrCb422ToYCbCr422_8u_C2P3R
4:2:2 CbYCr	2	<u>Cb0</u>	<u>Y0</u>	<u>Cr0</u>	<u>Y1</u>	<u>Cb1</u>	<u>Y2</u>	ippiCbYCr422ToRGB_8u_C2C3R
XYZ	3	<u>X0</u>	<u>Y0</u>	<u>Z0</u>	<u>X1</u>	<u>Y1</u>	<u>Z1</u>	ippiXYZToRGB_8u_C3R
XYZ	4	<u>X0</u>	<u>Y0</u>	<u>Z0</u>	<u>X1</u>	<u>Y1</u>	<u>Z1</u>	ippiXYZToRGB_16u_AC4R
LUV	3	<u>L0</u>	<u>U0</u>	<u>V0</u>	<u>L1</u>	<u>U1</u>	<u>V1</u>	ippiLUVToRGB_16s_C3R
LUV	4	<u>L0</u>	<u>U0</u>	<u>V0</u>	<u>A0</u>	<u>L1</u>	<u>U1</u>	ippiLUVToRGB_32f_AC4R
YCC	3	<u>Y0</u>	<u>C0</u>	<u>C0</u>	<u>Y1</u>	<u>C1</u>	<u>C1</u>	ippiYCCToRGB_8u_C3R
YCC	4	<u>Y0</u>	<u>C0</u>	<u>C0</u>	<u>A0</u>	<u>Y1</u>	<u>C1</u>	ippiYCCToRGB_8u_AC4R
HLS	3	<u>H0</u>	<u>L0</u>	<u>S0</u>	<u>H1</u>	<u>L1</u>	<u>S1</u>	ippiHLSToRGB_16u_C3R
HLS	4	<u>H0</u>	<u>L0</u>	<u>S0</u>	<u>A0</u>	<u>H1</u>	<u>L1</u>	ippiHLSToRGB_16u_AC4R
HSV	3	<u>H0</u>	<u>S0</u>	<u>V0</u>	<u>H1</u>	<u>S1</u>	<u>V1</u>	ippiHSVToRGB_16s_C3R
HSV	4	<u>H0</u>	<u>S0</u>	<u>V0</u>	<u>A0</u>	<u>H1</u>	<u>S1</u>	ippiHSVToRGB_16s_AC4R

Planar image formats supported by Intel IPP color conversion functions are listed in [Table 6-3](#) along with examples of Intel IPP functions using that format. Planes layout and their relative sizes are shown in [Figure 6-15](#) and [Figure 6-16](#).

**Table 6-3 Planar Image Formats**

Image Format	Number of Planes	Planes Layout				Example
		pl.1	pl.2	pl.3		
RGB	3	R	G	B	see <a href="#">Figure 6-15</a> , a	ippiRGBToYUV_8u_P3R
YUV	3	Y	U	V	see <a href="#">Figure 6-15</a> , a	ippiYUVToRGB_8u_P3R
4:2:2 YUV	3	Y	U	V	see <a href="#">Figure 6-15</a> , b	ippiYUV422ToRGB_8u_P3
4:2:0 YUV	3	Y	U	V	see <a href="#">Figure 6-15</a> , d	ippiYUV420ToRGB_8u_P3C3R
YCbCr	3	Y	Cb	Cr	see <a href="#">Figure 6-15</a> , a	ippiYCbCrToRGB_8u_P3R
4:2:2 YCbCr	3	Y	Cb	Cr	see <a href="#">Figure 6-15</a> , b	ippiYCbCr422_8u_P3C2R
4:1:1 YCbCr	3	Y	Cb	Cr	see <a href="#">Figure 6-15</a> , c	ippiYCbCr411_8u_P3P2R
4:1:1 YCbCr	2	Y	CbCr	-	see <a href="#">Figure 6-16</a> , a	ippiYCbCr411_8u_P2P3R
4:2:0 YCbCr	3	Y	Cb	Cr	see <a href="#">Figure 6-15</a> , d	ippiRGBToYCbCr420_8u_C3P3R
4:2:0 YCbCr	2	Y	CbCr	-	see <a href="#">Figure 6-16</a> , b	ippiYCbCr420_8u_P2P3R
4:2:0 YCrCb	3	Y	Cr	Cb	see <a href="#">Figure 6-15</a> , d	ippiYCrCb420ToYCbCr422_8u_P3R

**Figure 6-15 Plane Size and Layout - 3-planes Images**

---

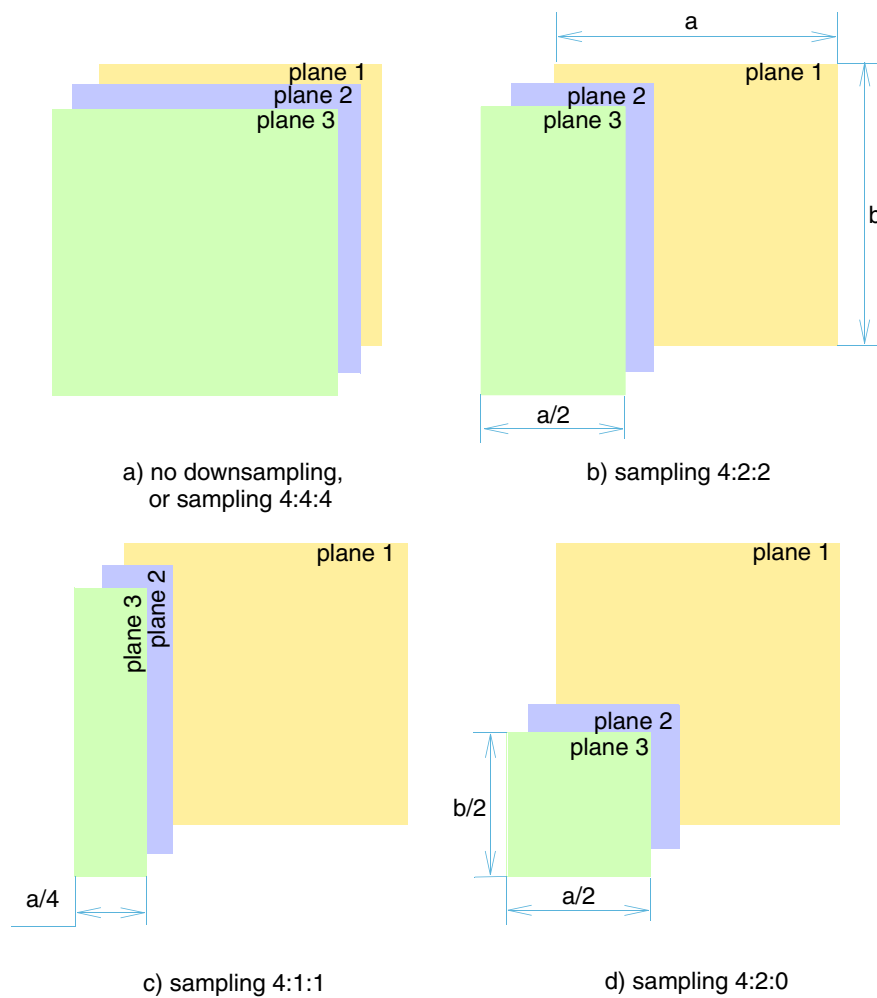
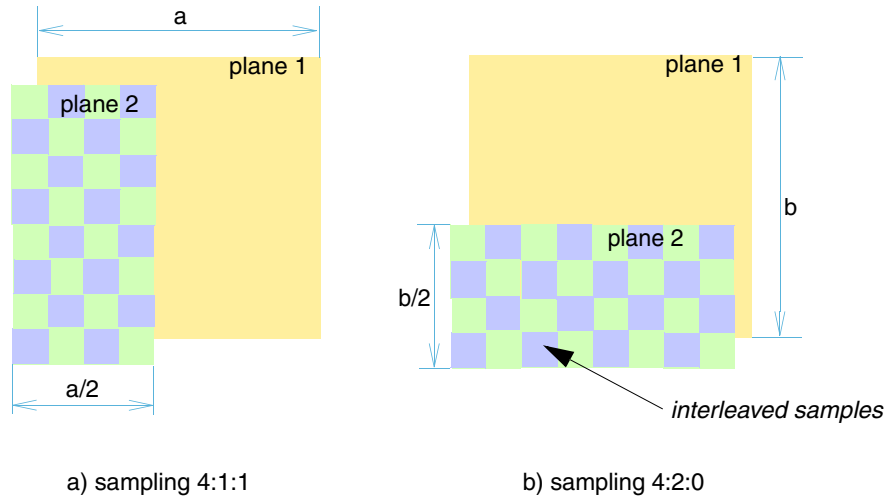


Figure 6-16 Plane Size and Layout - 2-planes Images



## Color Model Conversion

### RGBToYUV

*Converts an RGB image to the YUV color model.*

#### Syntax

##### Case 1: Operation on pixel-order data.

```
IppStatus ippRGBToYUV_<mod>(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C3R      8u\_AC4R



### Case 2: Operation on planar data.

```
IppStatus ippiRGBToYUV_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

### Case 3: Conversion from pixel-order to planar data.

```
IppStatus ippiRGBToYUV_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order data. An array of pointers to the source image ROI in separate color planes in case of planar data.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination ROI for pixel-order data. An array of pointers to destination buffers in separate color planes in case of planar data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRGBToYUV` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'U'V' image *pDst* according to the following formulas:

$$\begin{aligned}
 Y' &= 0.299 \cdot R' + 0.587 \cdot G' + 0.114 \cdot B' \\
 U' &= -0.147 \cdot R' - 0.289 \cdot G' + 0.436 \cdot B' = 0.492 \cdot (B' - Y') \\
 V' &= 0.615 \cdot R' - 0.515 \cdot G' - 0.100 \cdot B' = 0.877 \cdot (R' - Y')
 \end{aligned}$$

For digital RGB values in the range  $[0 \dots 255]$ ,  $Y'$  has the range  $[0 \dots 255]$ ,  $U$  varies in the range  $[-112 \dots +112]$ , and  $V$  in the range  $[-157 \dots +157]$ .

To fit in the range of  $[0 \dots 255]$ , a constant value 128 is added to computed  $U$  and  $V$  values, and  $V$  is then saturated.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## YUVToRGB

*Converts a YUV image to the RGB color model.*

---

### Syntax

#### Case 1: Operation on pixel-order data.

```
IppStatus ippiYUVToRGB_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

```
8u_C3R
8u_AC4R
```

#### Case 2: Operation on planar data.

```
IppStatus ippiYUVToRGB_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

#### Case 3: Conversion from planar-order to pixel-order data.

```
IppStatus ippiYUVToRGB_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Pointer to the source buffer for pixel-order data. An array of pointers to separate source color planes in case of planar data.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination buffer for pixel-order data. An array of pointers to separate destination color planes in case of planar data.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.

*roiSize*                      Size of the source and destination ROI in pixels.

## Description

The function `ippiYUVToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **Y'U'V'** image *pSrc* to the gamma-corrected R'G'B' image *pDst* according to the following formulas:

$$\begin{aligned} R' &= Y' + 1.140 * V' \\ G' &= Y' - 0.394 * U' - 0.581 * V' \\ B' &= Y' + 2.032 * U' \end{aligned}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## RGBToYUV422

*Converts an RGB image to the YUV color model;  
uses 4:2:2 sampling.*

---

### Syntax

#### Case 1: Operation on pixel-order data.

```
IppStatus ippiRGBToYUV422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Operation on planar data with ROI.

```
IppStatus ippiRGBToYUV422_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

#### Case 3: Operation on planar data without ROI.

```
IppStatus ippiRGBToYUV422_8u_P3(const Ipp8u* const pSrc[3],
    Ipp8u* pDst[3], IppiSize imgSize);
```

**Case 4: Conversion from pixel-order to planar data with ROI.**

```
IppStatus ippiRGBToYUV422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst[3], int dstStep[3], IppiSize roiSize);
```

**Case 5: Conversion from pixel-order to planar data without ROI.**

```
IppStatus ippiRGBToYUV422_8u_C3P3(const Ipp8u* pSrc, Ipp8u* pDst[3],
    IppiSize imgSize);
```

**Parameters**

<i>pSrc</i>	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffer in each color plane for planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image for operations with ROI.
<i>pDst</i>	Pointer to the destination image buffer for pixel-order image. An array of pointers to the destination image buffer in each color plane for planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI. An array of three values for planar image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>imgSize</i>	Size of the source and destination images in pixels for operations without ROI.

**Description**

The function `ippiRGBToYUV422` is declared in the `ippcc.h` file. This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'U'V' image *pDst* with [4:2:2 sampling](#) format, according to the same [formulas](#) as the function `ippiRGBToYUV` does. For more details on this sampling format, see [Table 6-2](#) and [Table 6-3](#).

For digital RGB values in the range  $[0..255]$ , Y' has the range  $[0..255]$ , U varies in the range  $[-112..+112]$ , and V in the range  $[-157..+157]$ .

To fit in the range of  $[0..255]$ , a constant value 128 is added to computed U and V values, and V is then saturated.

Some function flavors operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> or <code>imgSize</code> has a field with zero or negative value.

---

## YUV422ToRGB

*Converts a YUV image that has 4:2:2 sampling format to the RGB color model.*

---

### Syntax

#### Case 1: Operation on pixel-order data.

```
ippStatus ippiYUV422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Operation on planar data with ROI.

```
ippStatus ippiYUV422ToRGB_8u_P3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

#### Case 3: Operation on planar data without ROI.

```
ippStatus ippiYUV422ToRGB_8u_P3(const Ipp8u* const pSrc[3],
    Ipp8u* pDst[3], IppiSize imgSize);
```

#### Case 4: Conversion from planar-order to pixel-order data with ROI.

```
ippStatus ippiYUV422ToRGB_<mod>(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

```
8u_P3C3R    8u_P3AC4R
```

#### Case 5: Conversion from planar-order to pixel-order data without ROI.

```
ippStatus ippiYUV422ToRGB_8u_P3C3(const Ipp8u* const pSrc[3],
    Ipp8u* pDst, IppiSize imgSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffer in each color plane for planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image for operations with ROI. An array of three values in case of planar image.
<i>pDst</i>	Pointer to the destination image buffer for pixel-order image. An array of pointers to the destination image buffers in each color plane for planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>imgSize</i>	Size of the source and destination images in pixels for operations without ROI.

## Description

The function `ippiYUV422ToRGB` is declared in the `ippcc.h` file. This function converts the [Y'U'V'](#) image *pSrc* to the gamma-corrected R'G'B' image *pDst* according to the same [formulas](#) as the function `ippiYUVToRGB` does. The difference is that `ippiYUV422ToRGB` assumes the input data to be in [4:2:2 sampling](#) format (see [Table 6-2](#) and [Table 6-3](#) for more details).

The function `ppiYUV422ToRGB_P3AC4R` additionally creates an alpha channel in the destination image with alpha values set to zero.

Some function flavors operates with ROI (see [Regions of Interest in Intel IPP](#)).

The function flavors that do not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step arguments are not needed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>imgSize</i> has a field with zero or negative value.

## RGBToYUV420

*Converts an RGB image to the YUV color model;  
uses 4:2:0 sampling.*

---

### Syntax

#### Case 1: Operation on planar data with ROI.

```
IppStatus ippiRGBToYUV420_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

#### Case 2: Operation on planar data without ROI.

```
IppStatus ippiRGBToYUV420_8u_P3(const Ipp8u* const pSrc[3],
    Ipp8u* pDst[3], IppiSize imgSize);
```

#### Case 3: Conversion from pixel-order to planar data with ROI.

```
IppStatus ippiRGBToYUV420_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

#### Case 4: Conversion from pixel-order to planar data without ROI.

```
IppStatus ippiRGBToYUV420_8u_C3P3(const Ipp8u* pSrc, Ipp8u* pDst[3],
    IppiSize imgSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffers in each color plane for planar image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image for operations with ROI.
<i>pDst</i>	An array of pointers to the destination image buffers in each color plane.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in each plane of the destination image for operations with ROI.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>imgSize</i>	Size of the source and destination images in pixels for operations without ROI.

## Description

The function `ippiRGBToYUV420` is declared in the `ippcc.h` file. This function converts the gamma-corrected R'G'B' image `pSrc` to the Y'U'V' image `pDst` according to the same [formulas](#) as the function `ippiRGBToYUV` does. The difference is that `ippiRGBToYUV420` uses 4:2:0 sampling format for the converted image (see [Table 6-3](#) for more details).

For digital RGB values in the range  $[0 \dots 255]$ , Y' has the range  $[0 \dots 255]$ , U varies in the range  $[-112 \dots +112]$ , and V in the range  $[-157 \dots +157]$ .

To fit in the range of  $[0 \dots 255]$ , a constant value 128 is added to computed U and V values, and V is then saturated.

Some function flavors operates with ROI see [Regions of Interest in Intel IPP](#)).

The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> or <code>imgSize</code> has a field with zero or negative value.

---

## YUV420ToRGB

*Converts a YUV image that has 4:2:0 sampling format to the RGB image.*

---

## Syntax

### Case 1: Operation on planar data with ROI.

```
ippStatus ippiYUV420ToRGB_8u_P3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

### Case 2: Operation on planar data without ROI.

```
ippStatus ippiYUV420ToRGB_8u_P3(const Ipp8u* const pSrc[3],
    Ipp8u* pDst[3], IppiSize imgSize);
```



### Case 3: Conversion from planar-order to pixel-order data with ROI.

```
ippStatus ippiYUV420ToRGB_<mod>(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_P3C3R    8u_P3AC4R
```

### Case 4: Conversion from planar-order to pixel-order data without ROI.

```
ippStatus ippiYUV420ToRGB_8u_P3C3(const Ipp8u* const pSrc[3],
    Ipp8u* pDst, IppiSize imgSize);
```

## Parameters

<i>pSrc</i>	An array of pointers to the source image buffers in each color plane.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in each source image planes for operations with ROI.
<i>pDst</i>	Pointer to the destination image buffer for pixel-order images. An array of pointers to the destination image buffers in each color plane for planar images.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>imgSize</i>	Size of the source and destination images in pixels for operations without ROI.

## Description

The function `ippiYUV420ToRGB` is declared in the `ippcc.h` file. This function converts the [Y'U'V'](#) image *pSrc* to the gamma-corrected R'G'B' image *pDst* according to the same [formulas](#) as the function `ippiYUVToRGB` does. The difference is that `ippiYUV420ToRGB` assumes the input data to be in [4:2:0 sampling](#) format (see [Table 6-3](#) for more details).

The function `ippiYUV420ToRGB_P3AC4R` additionally creates an alpha channel in the destination image with alpha values set to zero.

Some function flavors operates with ROI see [Regions of Interest in Intel IPP](#)).

The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>imgSize</i> has a field with zero or negative value.

---

## YUV420ToBGR

*Converts a YUV image that has 4:2:0 sampling format to the BGR image.*

---

### Syntax

```
IppStatus ippYUV420ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],  
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	An array of pointers to ROI in each color plane in the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippYUV420ToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [Y'U'V'](#) image *pSrc* to the gamma-corrected B'G'R' image *pDst* according to the same [formulas](#) as the function `ippYUVToRGB` does. The function `ippYUV420ToBGR` assumes the input data to be in [4:2:0 sampling](#) format (see [Table 6-3](#) for more details).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> or <code>imgSize</code> has a field with zero or negative value.

---

## YUV420ToRGB565 YUV420ToRGB555 YUV420ToRGB444

*Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel RGB image.*

---

## Syntax

```

IppStatus ippiYUV420ToRGB565_8u16u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYUV420ToRGB555_8u16u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYUV420ToRGB444_8u16u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

```

## Parameters

<code>pSrc</code>	An array of pointers to ROI in each color plane in the source image.
<code>srcStep</code>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYUV420ToRGB444`, `ippiYUV420ToRGB555`, `ippiYUV420ToRGB565` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `y'u'v'` image `pSrc` to the gamma-corrected `R'G'B'` image `pDst` according to the same [formulas](#) as the function `ippiYUVToRGB` does. The difference is that the input data are in [4:2:0 sampling](#) format (see [Table 6-3](#) for more details). The destination image `pDst` is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see [Figure 6-14](#)): RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), or RGB444 (4 bits for red, green, blue). Bit reduction is performed by discarding the least significant bits in the image after color conversion.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## YUV420ToRGB565Dither YUV420ToRGB555Dither YUV420ToRGB444Dither

*Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel RGB image with dithering.*

---

### Syntax

```
IppStatus ippiYUV420ToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYUV420ToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYUV420ToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

`pSrc`                      An array of pointers to ROI in each color plane in the source image.

<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYUV420ToRGB565Dither`, `ippiYUV420ToRGB555Dither`, `ippiYUV420ToRGB444Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the [Y'U'V'](#) image *pSrc* to the gamma-corrected R'G'B' image *pDst* according to the same [formulas](#) as the function `ippiYUVToRGB` does. The difference is that the input data are in [4:2:0 sampling](#) format (see [Table 6-3](#) for more details). The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 6-14](#)): RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), or RGB444 (4 bits for red, green, blue). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [\[Thomas\]](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## BGR565ToYUV420 BGR555ToYUV420

*Convert 16-bit BGR image to the 4:2:0 YUV image.*

---

## Syntax

```
ippStatus ippiBGR565ToYUV420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
      Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippkBGR555ToYUV420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,  
      Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to ROI in the separate color planes in the destination image.
<i>dstStep</i>	Array of istances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippkBGR565ToYUV420`, `ippkBGR555ToYUV420` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the gamma-corrected B'G'R' image *pSrc* to the [Y'U'V'](#) image *pDst* according to the same [formulas](#) as the function `ippiRGBToYUV` does. The source image *pSrc* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of two possible formats (see [Figure 6-14](#)): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), and BGR555 (5 bits for blue, green, red). The destination image *pDst* has a [4:2:0 sampling](#) format (see [Table 6-3](#) for more details).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 0.

## YUV420ToBGR565

## YUV420ToBGR555

## YUV420ToBGR444

*Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel BGR image.*

---

### Syntax

```

IppStatus ippiYUV420ToBGR565_8u16u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYUV420ToBGR555_8u16u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYUV420ToBGR444_8u16u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

```

### Parameters

<i>pSrc</i>	An array of pointers to ROI in each separate color plane in the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYUV420ToBGR565`, `ippiYUV420ToBGR555`, `ippiYUV420ToBGR444` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the [Y'U'V'](#) image *pSrc* to the gamma-corrected B'G'R' image *pDst* according to the same [formulas](#) as the function `ippiYUVToRGB` does. The difference is that the input data are in [4:2:0 sampling](#) format (see [Table 6-3](#) for more details). The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 6-14](#)): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue,

green, red), or BGR444 (4 bits for blue, green, red).

Bit reduction is performed by discarding the least significant bits in the image after color conversion.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## YUV420ToBGR565Dither YUV420ToBGR555Dither YUV420ToBGR444Dither

*Convert a YUV image that has 4:2:0 sampling format to the 16-bit per pixel BGR image with dithering.*

---

### Syntax

```
IppStatus ippYUV420ToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippYUV420ToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippYUV420ToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	An array of pointers to ROI in each separate color plane in the source image.
<code>srcStep</code>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.



## Description

The functions `ippiYUV420ToBGR565Dither`, `ippiYUV420ToBGR555Dither`, `ippiYUV420ToBGR444Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the [Y'U'V'](#) image `pSrc` to the gamma-corrected B'G'R' image `pDst` according to the same [formulas](#) as the function `ippiYUVToRGB` does. The difference is that the input data are in [4:2:0 sampling](#) format (see [Table 6-3](#) for more details). The destination image `pDst` is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 6-14](#)): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), or BGR444 (4 bits for blue, green, red). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [[Thomas](#)].

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## RGBToYCbCr

*Converts an RGB image  
to the YCbCr color model.*

---

## Syntax

### Case 1: Operation on pixel-order data.

```
ippStatus ippiRGBToYCbCr_<mod>(const Ipp8u* pSrc, int srcStep,  
                                Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C3R`            `8u_AC4R`

### Case 2: Operation on planar data.

```
ippStatus ippiRGBToYCbCr_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,  
                                Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI. An array of pointers to ROI in the separate destination color planes for planar images.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRGBToYCbCr` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected R'G'B' image *pSrc* with values in the range [0..255] to the [Y'Cb'Cr'](#) image *pDst* according to the following formulas:

$$\begin{aligned} Y' &= 0.257 * R' + 0.504 * G' + 0.098 * B' + 16 \\ Cb' &= -0.148 * R' - 0.291 * G' + 0.439 * B' + 128 \\ Cr' &= 0.439 * R' - 0.368 * G' - 0.071 * B' + 128 \end{aligned}$$

In YCbCr model, Y is defined to have a nominal range [16..235], while Cb and Cr are defined to have a range [16..240], with 128 value as corresponding to zero.

Both the source and destination images have the same bit depth.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## YCbCrToRGB

*Converts a YCbCr image to the RGB color model.*

---

### Syntax

#### Case 1: Operation on pixel-order data.

```
IppStatus ippiYCbCrToRGB_<mod>(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C3R          8u\_AC4R

#### Case 2: Operation on planar data.

```
IppStatus ippiYCbCrToRGB_<mod>(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_P3R          8u\_P3C3R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI. An array of pointers to ROI in the separate destination color planes for planar images.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiYCbCrToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `'Y'Cb'Cr'` image *pSrc* to the 24-bit gamma-corrected `R'G'B'` image *pDst*. The following formulas are used for conversion:

$$\begin{aligned}
 R' &= 1.164 * (Y' - 16) + 1.596 * (Cr' - 128) \\
 G' &= 1.164 * (Y' - 16) - 0.813 * (Cr' - 128) - 0.392 * (Cb' - 128) \\
 B' &= 1.164 * (Y' - 16) + 2.017 * (Cb' - 128)
 \end{aligned}$$

The output  $R'G'B'$  values are saturated to the range  $[0..255]$ .

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## YCbCrToBGR

*Converts a YCbCr image to the BGR color model.*

---

### Syntax

```

IppStatus ippYCbCrToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippYCbCrToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

```

### Parameters

<code>pSrc</code>	An array of pointers to ROI in each separate source color planes.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>aval</code>	Constant value to create the fourth channel.

## Description

The function `ippiYCbCrToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'Cb'Cr'` image `pSrc` to the 24-bit gamma-corrected `B'G'R'` image `pDst` according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The output `B'G'R'` values are saturated to the range `[0..255]`.

The fourth channel is created by setting channel values to the constant value `aval`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 4 or <code>roiSize.height</code> is less than 1.

---

## YCbCrToRGB565 YCbCrToRGB555 YCbCrToRGB444

*Convert a YCbCr image  
to the 16-bit per pixel RGB image.*

---

## Syntax

### Case 1: Operation on pixel-order data.

```
ippStatus ippiYCbCrToRGB565_8u16u_C3R(const Ipp8u* pSrc, int srcStep,  
    Ipp16u* pDst, int dstStep, IppiSize roiSize);  
ippStatus ippiYCbCrToRGB555_8u16u_C3R(const Ipp8u* pSrc, int srcStep,  
    Ipp16u* pDst, int dstStep, IppiSize roiSize);  
ippStatus ippiYCbCrToRGB444_8u16u_C3R(const Ipp8u* pSrc, int srcStep,  
    Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from planar-order to pixel-order data.

```
ippStatus ippiYCbCrToRGB565_8u16u_P3C3R(const Ipp8u* pSrc[3],  
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```

IppStatus ippiYCbCrToRGB555_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB444_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);

```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYCbCrToRGB565`, `ippiYCbCrToRGB555`, `ippiYCbCrToRGB444` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `'Y'Cb'Cr'` image *pSrc* to the gamma-corrected `R'G'B'` image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 6-14](#) for more details): RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), or RGB444 (4 bits for red, green, blue). Bit reduction is performed by discarding the least significant bits in the image after color conversion.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## YCbCrToRGB565Dither

## YCbCrToRGB555Dither

## YCbCrToRGB444Dither

*Convert a YCbCr image  
to the 16-bit per pixel RGB image with dithering.*

---

### Syntax

#### Case 1: Operation on pixel-order data.

```
IppStatus ippiYCbCrToRGB565Dither_8u16u_C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB555Dither_8u16u_C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB444Dither_8u16u_C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from planar-order to pixel-order data.

```
IppStatus ippiYCbCrToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYCbCrToRGB565Dither`, `ippiYCbCrToRGB555Dither`, `ippiYCbCrToRGB444Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image `pSrc` to the gamma-corrected R'G'B' image `pDst` according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The destination image `pDst` is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 6-14](#) for more details): RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), or RGB444 (4 bits for red, green, blue). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [[Thomas](#)].

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## YCbCrToBGR565 YCbCrToBGR555 YCbCrToBGR444

*Convert a YCbCr image  
to the 16-bit per pixel BGR image.*

---

## Syntax

### Case 1: Operation on pixel-order data.

```

IppStatus ippiYCbCrToBGR565_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCrToBGR555_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCrToBGR444_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
    Ipp16u* pDst, int dstStep, IppiSize roiSize);

```



### Case 2: Conversion from planar-order to pixel-order data.

```
IppStatus ippiYCbCrToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR444_8u16u_P3C3R(const Ipp8u* pSrc[3], int
    srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCrToBGR565`, `ippiYCbCrToBGR555`, `ippiYCbCrToBGR444` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc* to the gamma-corrected `B'G'R'` image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 6-14](#)): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), or BGR444 (4 bits for blue, green, red).

Bit reduction is performed by discarding the least significant bits in the image after color conversion.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## YCbCrToBGR565Dither

## YCbCrToBGR555Dither

## YCbCrToBGR444Dither

*Convert a YCbCr image  
to the 16-bit per pixel BGR image with dithering.*

### Syntax

#### Case 1: Operation on pixel-order data.

```
IppStatus ippiYCbCrToBGR565Dither_8u16u_C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR555Dither_8u16u_C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR444Dither_8u16u_C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from planar-order to pixel-order data.

```
IppStatus ippiYCbCrToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYCbCrToBGR565Dither`, `ippiYCbCrToBGR555Dither`, `ippiYCbCrToBGR444Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image `pSrc` to the gamma-corrected B'G'R' image `pDst` according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The destination image `pDst` is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 6-14](#) for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), or BGR444 (4 bits for blue, green, red). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [[Thomas](#)].

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## RGBToYCbCr422

*Converts 24-bit per pixel RGB image  
to 16-bit per pixel YCbCr image*

---

## Syntax

### Case 1: Operation on pixel-order data.

```
IppStatus ippiRGBToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,  
                                     Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from planar-order to pixel-order data.

```
IppStatus ippiRGBToYCbCr422_8u_P3C2R(const Ipp8u* const pSrc[3],  
                                     int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRGBToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'Cb'Cr' image *pDst* according to the same [formulas](#) as the function `ippiRGBToYCbCr` does. The difference is that `ippiRGBToYCbCr422` uses [4:2:2 sampling](#) format for the converted image (see [Table 6-2](#) and [Table 6-3](#) for more details).

The converted buffer has the reduced bit depth of 16 bits per pixel, whereas the source buffer has 24 bit depth.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## YCbCr422ToRGB

*Converts 16-bit per pixel YCbCr image to 24-bit per pixel RGB image.*

---

## Syntax

### Case 1: Operation on pixel-order data.

```
ippStatus ippiYCbCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from pixel-order to planar data.

```
IppStatus ippiYCbCr422ToRGB_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

### Case 3: Conversion from planar-order to pixel-order data.

```
IppStatus ippiYCbCr422ToRGB_8u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiYCbCr422ToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `'Y'Cb'Cr'` image *pSrc*, packed in [4:2:2 sampling](#) format (see [Table 6-2](#) and [Table 6-3](#) for more details) to the 24-bit gamma-corrected `'R'G'B'` image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The output `'R'G'B'` values are saturated to the range `[0..255]`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## BGRToYCbCr422

*Converts 24-bit per pixel BGR image  
to 16-bit per pixel YCbCr image.*

---

### Syntax

#### Case 1: Operation on pixel-order data.

```
IppStatus ippiBGRToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToYCbCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from pixel-order to planar data.

```
IppStatus ippiBGRToYCbCr422_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCbCr422_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each plane of the destination planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination pixel-order image. An array of distances in bytes for each plane of the destination planar image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiBGRToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channels gamma-corrected B'G'R' image *pSrc* to the two-channels or three-planes Y'Cb'Cr' image *pDst* according to the same [formulas](#) as the function `ippiRGBToYCbCr` does. The difference is that `ippiBGRToYCbCr422` uses [4:2:2 sampling](#) format (see [Table 6-2](#) and [Table 6-3](#) for more details).

### Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippiStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 1.

---

## YCbCr422ToBGR

*Converts 16-bit per pixel YCbCr image to 24-bit per pixel BGR image.*

---

### Syntax

```

IppStatus ippiYCbCr422ToBGR_8u_C2C3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

### Description

The function `ippiYCbCr422ToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'Cb'Cr'` image `pSrc`, packed in [4:2:2 sampling](#) format (see [Table 6-2](#) and [Table 6-3](#) for more details) to the 24-bit gamma-corrected `B'G'R'` image `pDst` according to the same [formulas](#) as the function `ippiYCbCrToRGB` does. The output `B'G'R'` values are saturated to the range `[0..255]`.

The fourth channel is created by setting channel values to the constant value `aval`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2 or <code>roiSize.height</code> is less than 1.

---

## BGR565ToYCbCr422

## BGR555ToYCbCr422

*Converts 16-bit per pixel BGR image  
to 16-bit per pixel YCbCr image.*

---

### Syntax

#### Case 1: Operation on pixel-order data.

```
ippStatus ippiBGR565ToYCbCr422_16u8u_C3C2R(const Ipp16u* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
ippStatus ippiBGR555ToYCbCr422_16u8u_C3C2R(const Ipp16u* pSrc, int srcStep,
      Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from pixel-order to planar data.

```
ippStatus ippiBGR565ToYCbCr422_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
      Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
ippStatus ippiBGR555ToYCbCr422_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
      Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

`pSrc`                      Pointer to the source image ROI.



<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination pixel-order image. An array of distances in bytes for each plane of the destination planar image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiBGR565ToYCbCr422` and `ippiBGR555ToYCbCr422` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a three- or four-channels gamma-corrected B'G'R' image *pSrc* to the two-channels or three-planes Y'Cb'Cr' image *pDst* according to the same [formulas](#) as the function `ippiRGBToYCbCr` does. The difference is that these functions use [4:2:2 sampling](#) format (see [Table 6-2](#) and [Table 6-3](#) for more details).

The source image *pSrc* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of two possible formats (see [Figure 6-14](#) for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), and BGR555 (5 bits for blue, green, red).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 1.

## RGBToCbYCr422

### RGBToCbYCr422Gamma

*Convert 24-bit per pixel RGB image  
to 16-bit per pixel CbYCr image.*

#### Syntax

```
IppStatus ippiRGBToCbYCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiRGBToCbYCr422Gamma_8u_C3C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

#### Description

The functions `ippiRGBToCbYCr422` and `ippiRGBToCbYCr422Gamma` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

The function `ippiRGBToCbYCr422` converts the gamma-corrected R'G'B' image *pSrc* to the Cb'Y'Cr' image *pDst* according to the same [formulas](#) as the function `ippiRGBToYCbCr` does. The function `ippiRGBToCbYCr422Gamma` performs gamma-correction of the source RGB image *pSrc* according to the same [formula](#) as the function `ippiGammaFwd` does, and then converts it to the Cb'Y'Cr' image *pDst* according to the same [formulas](#) as the function `ippiRGBToYCbCr` does.

The functions `ippiRGBToCbYCr422` and `ippiRGBToCbYCr422Gamma` use [4:2:2 sampling](#) format for the converted image.

An CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ....

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## CbYCr422ToRGB

*Converts 16-bit per pixel CbYCr image to 24-bit per pixel RGB image.*

---

### Syntax

```
IppStatus ippCbYCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep,
                                     Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels

### Description

The function `ippCbYCr422ToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **Cb'Y'Cr'** image *pSrc*, packed in [4:2:2 sampling](#) format, to the 24-bit gamma-corrected **R'G'B'** image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

A CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ... .

The output **R'G'B'** values are saturated to the range `[0..255]`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## BGRToCbYCr422

*Converts 32-bit per pixel BGR image to 16-bit per pixel CbYCr image.*

---

### Syntax

```
IppStatus ippibGRToCbYCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippibGRToCbYCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the four-channel gamma-corrected B'G'R' image *pSrc* to the two-channel Cb'Y'Cr' image *pDst* according to the same [formulas](#) as the function `ippiRGBToYCbCr` does. The function `ippibGRToCbYCr422` uses [4:2:2 sampling](#) format for the converted image. An alpha-channel information is lost.

An CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ... .

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## CbYCr422ToBGR

*Converts 16-bit per pixel CbYCr image to four channel BGR image.*

---

### Syntax

```
IppStatus ippCbYCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep,
                                     Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

### Description

The function `ippCbYCr422ToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the **Cb'Y'Cr'** image *pSrc*, packed in [4:2:2 sampling](#) format, to the four channel gamma-corrected **B'G'R'** image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

A CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ... .

The output **B'G'R'** values are saturated to the range `[0..255]`.

The fourth channel is created by setting channel values to the constant value *aval*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## YCbCr422ToRGB565 YCbCr422ToRGB555 YCbCr422ToRGB444

*Convert 16-bit per pixel YCbCr image  
to 16-bit per pixel RGB image.*

---

### Syntax

#### Case 1: Operation on pixel-order data.

```
IppStatus ippYCbCr422ToRGB565_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippYCbCr422ToRGB555_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippYCbCr422ToRGB444_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from planar-order to pixel-order data .

```
IppStatus ippYCbCr422ToRGB565_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippYCbCr422ToRGB555_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippYCbCr422ToRGB444_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order images. An array of pointers to ROI in each separate source planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. An array of three values in case of planar data.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYCbCr422ToRGB565`, `ippiYCbCr422ToRGB565`, `ippiYCbCr422ToRGB565` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc*, packed in [4:2:2 sampling](#) format (see [Table 6-2](#) for more details), to the gamma-corrected `R'G'B'` image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see [Figure 6-14](#)): `RGB565` (5 bits for red, 6 bits for green, 5 bits for blue), `RGB555` (5 bits for red, green, blue), `RGB444` (4 bits for red, green, blue).

Bit reduction is performed by discarding the least significant bits in the image after color conversion.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## YCbCr422ToRGB565Dither

## YCbCr422ToRGB555Dither

## YCbCr422ToRGB444Dither

*Convert 16-bit per pixel YCbCr image  
to 16-bit per pixel RGB image with dithering.*

### Syntax

#### Case 1: Operation on pixel-order data.

```
IppStatus ippiYCbCr422ToRGB565Dither_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB555Dither_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB444Dither_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from planar-order to pixel-order data .

```
IppStatus ippiYCbCr422ToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order images. An array of pointers to ROI in each separate source planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. An array of three values in case of planar data.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.



## Description

The functions `ippiYCbCr422ToRGB565Dither`, `ippiYCbCr422ToRGB565Dither`, `ippiYCbCr422ToRGB565Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `'Y'Cb'Cr'` image `pSrc`, packed in [4:2:2 sampling](#) format (see [Table 6-2](#) for more details), to the gamma-corrected `R'G'B'` image `pDst` according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The destination image `pDst` is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of three possible formats (see [Figure 6-14](#)): RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), RGB444 (4 bits for red, green, blue). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [[Thomas](#)].

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## YCbCr422ToBGR565 YCbCr422ToBGR555 YCbCr422ToBGR444

*Convert 16-bit per pixel YCbCr image  
to 16-bit per pixel BGR image.*

---

## Syntax

### Case 1: Operation on pixel-order data.

```
ippStatus ippiYCbCr422ToBGR565_8u16u_C2C3R(const Ipp8u* pSrc,
      int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
ippStatus ippiYCbCr422ToBGR555_8u16u_C2C3R(const Ipp8u* pSrc,
      int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToBGR444_8u16u_C2C3R(const Ipp8u* pSrc,
int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Case 2: Conversion from planar-order to pixel-order data .

```
IppStatus ippiYCbCr422ToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3],
int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3],
int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR444_8u16u_P3C3R(const Ipp8u* pSrc[3],
int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order images. An array of pointers to ROI in each separate source planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. An array of three values in case of planar data.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCr422ToBGR565`, `ippiYCbCr422ToBGR565`, `ippiYCbCr422ToBGR555` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc*, packed in [4:2:2 sampling](#) format (see [Table 6-2](#) for more details), to the gamma-corrected `B'G'R'` image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see [Figure 6-14](#)): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), BGR444 (4 bits for blue, green, red).

Bit reduction is performed by discarding the least significant bits in the image after color conversion.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## YCbC422ToBGR565Dither YCbC422ToBGR555Dither YCbCr422ToBGR444Dither

*Convert 16-bit per pixel YCbCr image  
to 16-bit per pixel BGR image with dithering.*

---

### Syntax

#### Case 1: Operation on pixel-order data.

```
ippStatus ippYCbCr422ToBGR565Dither_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
ippStatus ippYCbCr422ToBGR555Dither_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
ippStatus ippYCbCr422ToBGR444Dither_8u16u_C2C3R(const Ipp8u* pSrc,
    int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: Conversion from planar-order to pixel-order data.

```
ippStatus ippYCbCr422ToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
ippStatus ippYCbCr422ToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
ippStatus ippYCbCr422ToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI for pixel-order images. An array of pointers to ROI in each separate source planes for planar images.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. An array of three values in case of planar data.
<i>pDst</i>	Pointer to the destination image ROI.

---

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYCbCr422ToBGR565Dither`, `ippiYCbCr422ToBGR565Dither`, `ippiYCbCr422ToBGR565Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc*, packed in [4:2:2 sampling](#) format (see [Table 6-2](#) for more details), to the gamma-corrected `B'G'R'` image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see [Figure 6-14](#)): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), BGR444 (4 bits for blue, green, red). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [[Thomas](#)].

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## RGBToYCbCr420

*Converts an RGB image  
to the YCbCr color model;  
uses 4:2:0 sampling.*

---

## Syntax

```
IppStatus ippiRGBToYCbCr420_8u_C3P3R(const Ipp8u* pSrc, int srcStep,  
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in the separate planes of the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRGBToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the gamma-corrected R'G'B' image *pSrc* to the Y'Cb'Cr' image *pDst* according to the same [formulas](#) as the function `ippiRGBToYCbCr` does. The difference is that `ippiRGBToYCbCr420` uses [4:2:0 sampling](#) format for the converted image (see [Table 6-3](#) for more details).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## YCbCr420ToRGB

*Converts a YCbCr image that has 4:2:0 sampling format to the RGB color model.*

---

## Syntax

```
IppStatus ippiYCbCr420ToRGB_8u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiYCbCr420ToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the `Y'Cb'Cr'` image *pSrc* to the gamma-corrected `R'G'B'` image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does. The difference is that `ippiYCbCr420ToRGB` assumes the input data to be in [4:2:0 sampling](#) format, in which the number of Cb and Cr samples is reduced by half in both vertical and horizontal directions (see [Table 6-3](#) for more details).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## YCbCr420ToRGB565

## YCbCr420ToRGB555

## YCbCr420ToRGB444

*Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel RGB image.*

---

## Syntax

```

IppStatus ippiYCbCr420ToRGB565_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToRGB555_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToRGB444_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

```

## Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

The functions `ippiYCbCr420ToRGB565`, `ippiYCbCr422ToRGB565`, `ippiYCbCr422ToRGB565` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc*, packed in [4:2:0 sampling](#) format (see [Table 6-2](#) for more details), to the gamma-corrected `R'G'B'` image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see [Figure 6-14](#)): RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), RGB444 (4 bits for red, green, blue).

Bit reduction is performed by discarding the least significant bits in the image after color conversion.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## YCbCr420ToRGB565Dither

## YCbCr420ToRGB555Dither

## YCbCr420ToRGB444Dither

*Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel RGB image with dithering.*

### Syntax

```
IppStatus ippiYCbCr420ToRGB565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToRGB555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToRGB444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCr420ToRGB565Dither`, `ippiYCbCr420ToRGB555Dither`, `ippiYCbCr420ToRGB444Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc*, packed in [4:2:0 sampling](#) format (see [Table 6-2](#) for more details), to the gamma-corrected `R'G'B'` image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit RGB image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of



three possible formats (see [Figure 6-14](#) for more details): RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), RGB555 (5 bits for red, green, blue), or RGB444 (4 bits for red, green, blue). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [[Thomas](#)].

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## BGRToYCbCr420

*Converts a BGR image to the YCbCr image with 4:2:0 sampling format.*

---

### Syntax

```
IppStatus ippIBGRToYCbCr420_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippIBGRToYCbCr420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	An array of pointers to ROI in separate planes of the destination image.
<code>dstStep</code>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippIBGRToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channels gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* according to the same [formulas](#) as the function `ippiRGBToYCbCr` does. The difference is that `ippiBGRToYCbCr420` uses [4:2:0 sampling](#) format (see [Table 6-3](#) for more details).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 2.

---

## YCbCr420ToBGR

*Converts a YCbCr image that has 4:2:0 sampling format to the BGR color model.*

---

### Syntax

```
IppStatus ippiYCbCr420ToBGR_8u_P3C3R(const Ipp8u* const pSrc[3],
    int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiYCbCr420ToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [Y'Cb'Cr'](#) image *pSrc* to the gamma-corrected B'G'R' image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does. The difference is that `ippiYCbCr420ToBGR` assumes the input data to be in [4:2:0 sampling](#) format, in which the number of Cb and Cr samples is reduced by half in both vertical and horizontal directions (see [Table 6-3](#) for more details).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## BGR565ToYCbCr420, BGR555ToYCbCr420

*Converts a 16-bit per pixel BGR image to the YCbCr image that has 4:2:0 sampling format.*

---

### Syntax

```
IppStatus ippiBGR565ToYCbCr420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
      Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGR555ToYCbCr420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
      Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiBGR565ToYCbCr420` and `ippiBGR555ToYCbCr420` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a three-channels gamma-corrected B'G'R' image `pSrc` to the planar Y'Cb'Cr' image `pDst` according to the same [formulas](#) as the function `ippiRGBToYCbCr` does. The destination image has a [4:2:0 sampling](#) format (see [Table 6-3](#) for more details).

The source image `pSrc` is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of two possible formats (see [Figure 6-14](#) for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), and BGR555 (5 bits for blue, green, red).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2 or <code>roiSize.height</code> is less than 2.

---

## YCbCr420ToBGR565

## YCbCr420ToBGR555

## YCbCr420ToBGR444

*Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel BGR image.*

---

## Syntax

```
IppStatus ippiYCbCr420ToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3],
      int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3],
      int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr420ToBGR444_8u16u_P3C3R(const Ipp8u* pSrc[3],
      int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

`pSrc`                      An array of pointers to ROI in separate planes of the source image.

<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYCbCr420ToBGR565`, `ippiYCbCr420ToBGR565`, `ippiYCbCr420ToBGR565` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc*, packed in [4:2:0 sampling](#) format (see [Table 6-2](#) for more details), to the gamma-corrected `B'G'R'` image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of three possible formats (see [Figure 6-14](#)): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), BGR444 (4 bits for blue, green, red).

Bit reduction is performed by discarding the least significant bits in the image after color conversion.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## YCbCr420ToBGR565Dither

## YCbCr420ToBGR555Dither

## YCbCr420ToBGR444Dither

*Convert a YCbCr image that has 4:2:0 sampling format to the 16-bit per pixel BGR image with dithering.*

### Syntax

```

IppStatus ippiYCbCr420ToBGR565Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToBGR555Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToBGR444Dither_8u16u_P3C3R(const Ipp8u* pSrc[3],
    int srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);

```

### Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCr420ToBGR565Dither`, `ippiYCbCr420ToBGR555Dither`, `ippiYCbCr420ToBGR444Dither` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the `Y'Cb'Cr'` image *pSrc*, packed in [4:2:0 sampling](#) format (see [Table 6-2](#) for more details), to the gamma-corrected `B'G'R'` image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of

three possible formats (see [Figure 6-14](#) for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), BGR555 (5 bits for blue, green, red), BGR444 (4 bits for blue, green, red). Bit reduction is performed using the Bayer's threshold dithering algorithm with 4x4 matrix [[Thomas](#)].

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## BGRToYCrCb420

*Converts a BGR image to the YCrCb image with 4:2:0 sampling format.*

---

### Syntax

```

IppStatus ippIBGRToYCrCb420_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippIBGRToYCrCb420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	An array of pointers to ROI in separate planes of the destination image.
<code>dstStep</code>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippIBGRToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channels gamma-corrected B'G'R' image *pSrc* to the planar Y'Cr'Cb' image *pDst* according to the same [formulas](#) as the function `ippiRGBToYCrCbCr` does. The destination image *pDst* has [4:2:0 sampling](#) format and the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table 6-3](#) for more details).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 2.

---

## BGR565ToYCrCb420 BGR555ToYCrCb420

*Converts a 16-bit per pixel BGR image  
to the YCrCb image with 4:2:0 sampling format.*

---

### Syntax

```
IppStatus ippiBGR565ToYCrCb420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,  
      Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);  
IppStatus ippiBGR555ToYCrCb420_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,  
      Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.



## Description

The functions `ippiBGR565ToYCrCb420` and `ippiBGR555ToYCrCb420` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a three-channels gamma-corrected B'G'R' image `pSrc` to the planar Y'Cr'Cb' image `pDst` according to the same [formulas](#) as the function `ippiRGBToYCbCr` does. The destination image `pDst` has [4:2:0 sampling](#) format and the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table 6-3](#) for more details).

The source image `pSrc` is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of two possible formats (see [Figure 6-14](#) for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), and BGR555 (5 bits for blue, green, red).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> is less than 2 or <code>roiSize.height</code> is less than 2.

---

## BGRToYCbCr411

*Converts a BGR image to the YCbCr planar image that has a 4:1:1 sampling format.*

---

## Syntax

```

IppStatus ippiBGRToYCbCr411_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCbCr411_8u_AC4P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

```

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	An array of pointers to ROI in separate planes of the destination image.

<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRTToYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a three- or four-channels gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* according to the same [formulas](#) as the function `ippiRGBToYCbCr` does. The difference is that `ippiBGRTToYCbCr411` uses [4:1:1 sampling](#) format (see [Table 6-3](#) for more details).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 1.

---

## YCbCr411ToBGR

*Converts a YCbCr image that has 4:1:1 sampling format to the RGB color model.*

---

## Syntax

```
IppStatus ippiYCbCr411ToBGR_8u_P3C3R(const Ipp8u* pSrc[3],
                                     int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr411ToBGR_8u_P3C4R(const Ipp8u* pSrc[3],
                                     int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize,
                                     Ipp8u aval);
```

## Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create fourth channel.

## Description

The function `ippiYCbCr411ToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the planar `Y'Cb'Cr'` image *pSrc* to the three- or four-channels image *pDst*. To compute gamma-corrected R'G'B' (B'G'R') channel values the above [formulas](#) are used. The difference is that `ippiYCbCr411ToBGR` assumes the input data to be in [4:1:1 sampling](#) format (see [Table 6-3](#) for more details). Fourth channel is created by setting channel values to the constant value *aval*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## BGR565ToYCbCr411, BGR555ToYCbCr411

*Converts a 16-bit per pixel BGR image  
to the YCbCr image that has 4:1:1 sampling format.*

---

## Syntax

```
ippStatus ippiBGR565ToYCbCr411_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
      Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

ippStatus ippiBGR555ToYCbCr411_16u8u_C3P3R(const Ipp16u* pSrc, int srcStep,
      Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	An array of pointers to ROI in separate planes of the destination image.
<i>dstStep</i>	An array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiBGR565ToYCbCr411` and `ippiBGR555ToYCbCr411` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a three-channels gamma-corrected B'G'R' image *pSrc* to the planar Y'Cb'Cr' image *pDst* according to the same [formulas](#) as the function `ippiRGBToYCbCr` does. The difference is that these functions use [4:1:1 sampling](#) format (see [Table 6-3](#) for more details).

The source image *pSrc* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (`16u` data type). It can be in one of two possible formats (see [Figure 6-14](#) for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), and BGR555 (5 bits for blue, green, red).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 1.

---

## YCbCr411ToBGR565, YCbCr411ToBGR555

*Convert a YCbCr image that has 4:1:1 sampling format to the 16-bit per pixel BGR image.*

---

### Syntax

```
IppStatus ippiYCbCr411ToBGR565_8u16u_P3C3R(const Ipp8u* pSrc[3], int  
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);  
  
IppStatus ippiYCbCr411ToBGR555_8u16u_P3C3R(const Ipp8u* pSrc[3], int  
    srcStep[3], Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	An array of pointers to ROI in separate planes of the source image.
<i>srcStep</i>	An array of distances in bytes between starts of consecutive lines in the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCr411ToBGR565` and `ippiYCbCr411ToBGR555` are declared in the `ippcc.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert the planar `Y'Cb'Cr'` image *pSrc*, packed in [4:1:1 sampling](#) format (see [Table 6-3](#) for more details), to the gamma-corrected `B'G'R'` image *pDst* according to the same [formulas](#) as the function `ippiYCbCrToRGB` does.

The destination image *pDst* is a packed 16-bit BGR image with reduced bit depth; all 3 channel intensities for a pixel are packed into two consecutive bytes (16u data type). It can be in one of two possible formats (see [Figure 6-14](#) for more details): BGR565 (5 bits for blue, 6 bits for green, 5 bits for red) and BGR555 (5 bits for blue, green, red).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

---

<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## RGBToXYZ

*Converts an RGB image to the XYZ color model.*

---

### Syntax

```
IppStatus ippRGBToXYZ_<mod>(const Ipp<datatype>* pSrc, int srcStep,
                             Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippRGBToXYZ` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the RGB image `pSrc` to the CIE [XYZ](#) image `pDst` according to the following basic equations:

$$\begin{aligned} X &= 0.412453 * R + 0.35758 * G + 0.180423 * B \\ Y &= 0.212671 * R + 0.71516 * G + 0.072169 * B \\ Z &= 0.019334 * R + 0.119193 * G + 0.950227 * B \end{aligned}$$

The equations above are given on the assumption that R, G, and B values are normalized to the

range `[0..1]`. In case of the floating-point data type, the input RGB values must already be in the range `[0..1]`. For integer data types, normalization is done by the conversion function internally. The computed XYZ values are saturated if they fall out of range `[0..1]`. In case of integer function flavors, these values are then scaled to the full range of the destination data type (see [Table 2-2](#) in Chapter 2).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## XYZToRGB

*Converts an XYZ image to the RGB color model.*

---

### Syntax

```
IppStatus ippXYZToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep,
                             Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

The function `ippiXYZToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the CIE [XYZ](#) image *pSrc* to the RGB image *pDst* according to the following basic equations:

$$\begin{aligned} R &= 3.240479 * X - 1.53715 * Y - 0.498535 * Z \\ G &= -0.969256 * X + 1.875991 * Y + 0.041556 * Z \\ B &= 0.055648 * X - 0.204043 * Y + 1.057311 * Z \end{aligned}$$

The equations above are given on the assumption that *X*, *Y*, and *Z* values are in the range  $[0..1]$ . In case of the floating-point data type, the input XYZ values must already be in the range  $[0..1]$ . For integer data types, normalization is done by the conversion function internally.

The computed RGB values are saturated if they fall out of range  $[0..1]$ .

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see [Table 2-2](#) in Chapter 2).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## RGBToLUV

*Converts an RGB image to the LUV color model.*

---

## Syntax

```
ippStatus ippiRGBToLUV_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>



## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRGBToLUV` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the RGB image *pSrc* to the [CIE LUV](#) image *pDst* in two steps. First, the conversion is done into [CIE XYZ](#) format, using equations defined for the `ippiRGBToXYZ` function. After that, conversion to LUV image is performed in accordance with the following equations:

$$L = 116. \cdot (Y/Y_n)^{1/3} - 16.$$

$$U = 13. \cdot L \cdot (u - u_n)$$

$$V = 13. \cdot L \cdot (v - v_n)$$

where

$$u = 4. \cdot X / (X + 15. \cdot Y + 3. \cdot Z)$$

$$v = 9. \cdot Y / (X + 15. \cdot Y + 3. \cdot Z)$$

$$u_n = 4. \cdot x_n / (-2. \cdot x_n + 12. \cdot y_n + 3. \cdot )$$

$$v_n = 9. \cdot y_n / (-2. \cdot x_n + 12. \cdot y_n + 3. \cdot )$$

Here  $x_n = 0.312713$ ,  $y_n = 0.329016$  are the CIE chromaticity coordinates of the D65 white point, and  $y_n = 1.0$  is the luminance of the D65 white point.

The computed values of the L component are in the range  $[0..100]$ , U component in the range  $[-134..220]$ , and V component in the range  $[-140..122]$ .

The equations above are given on the assumption that R, G, and B values are normalized to the range  $[0..1]$ . In case of the floating-point data type, the input RGB values must already be in the range  $[0..1]$ . For integer data types, normalization is done by the conversion function internally. In case of 8u data type, the computed L, U, and V values are quantized and converted to fit in the range  $[0..IPP\_MAX\_8U]$  as follows:

```

L = L * IPP_MAX_8U / 100.
U = (U + 134.) * IPP_MAX_8U / 354.
V = (V + 140.) * IPP_MAX_8U / 256.

```

In case of `16u` data type, the computed `L`, `U`, and `V` values are quantized and converted to fit in the range `[0..IPP_MAX_16U]` as follows:

```

L = L * IPP_MAX_16U / 100.
U = (U + 134.) * IPP_MAX_16U / 354.
V = (V + 140.) * IPP_MAX_16U / 256.

```

In case of `16s` data type, the computed `L`, `U`, and `V` values are quantized and converted to fit in the range `[IPP_MIN_16S..IPP_MAX_16S]` as follows:

```

L = L * IPP_MAX_16U / 100. + IPP_MIN_16S
U = (U + 134.) * IPP_MAX_16U / 354. + IPP_MIN_16S
V = (V + 140.) * IPP_MAX_16U / 256. + IPP_MIN_16S

```

For `32f` data type, no further conversion is done and `L`, `U`, and `V` components remain in the ranges `[0..100]`, `[-134..220]`, and `[-140..122]`, respectively.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## LUVToRGB

*Converts a LUV image to the RGB color model.*

---

### Syntax

```

IppStatus ippiLUVToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

```

Supported values for `mod`:

```

8u_C3R      16u_C3R      16s_C3R      32f_C3R

```

8u\_AC4R      16u\_AC4R      16s\_AC4R      32f\_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiLUVToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [CIE LUV](#) image *pSrc* to the RGB image *pDst* in two steps. First, the conversion is carried out into [CIE XYZ](#) format.

To accomplish it, LUV components are transformed back into their original range. This is done for different data types in the following way.

For 8u data type:

```
L = L * 100. / IPP_MAX_8U
U = (U * 354. / IPP_MAX_8U) - 134.
V = (V * 256. / IPP_MAX_8U) - 140.
```

For 16u data type:

```
L = L * 100. / IPP_MAX_16U
U = (U * 354. / IPP_MAX_16U) - 134.
V = (V * 256. / IPP_MAX_16U) - 140.
```

For 16s data type:

```
L = (L - IPP_MIN_16S) * 100. / IPP_MAX_16U
U = ((U - IPP_MIN_16S) * 354. / IPP_MAX_16U) - 134.
V = ((V - IPP_MIN_16S) * 256. / IPP_MAX_16U) - 140.
```

After that, conversion to XYZ format takes place as follows:

```
Y = Yn * ((L + 16.) / 116.)**3.
X = -9.* Y * u / ((u - 4.)* v - u * v)
Z = (9.* Y - 15*v*Y - v*X) / 3. * v
```

where

$$u = U / (13. * L) + u_n$$

$$v = V / (13. * L) + v_n$$

and

$$u_n = 4. * x_n / (-2. * x_n + 12. * y_n + 3.)$$

$$v_n = 9. * y_n / (-2. * x_n + 12. * y_n + 3.)$$

Here  $x_n = 0.312713$ ,  $y_n = 0.329016$  are the CIE chromaticity coordinates of the D65 white point, and  $y_n = 1.0$  is the luminance of the D65 white point.

After this intermediate conversion is done, the obtained XYZ image is then converted to the destination RGB format using equations defined for the [ippiXYZToRGB](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## BGRToLab

*Converts a BGR image to the Lab color model.*

---

### Syntax

```
ippStatus ippiBGRToLab_8u_C3R(const Ipp8u* pSrc, int srcStep,
                               Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
ippStatus ippiBGRToLab_8u16u_C3R(const Ipp8u* pSrc, int srcStep,
                                   Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

*roiSize*                      Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRTToLab` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the BGR image *pSrc* to the [CIE Lab](#) image *pDst* in two steps. First, the conversion is done into [CIE XYZ](#) format, using equations defined for the [ippiRGBToXYZ](#) function. After that, conversion to Lab image is performed in accordance with the following equations:

$$\begin{aligned} L &= 116. \cdot (Y/Y_n)^{1/3} - 16 && \text{for } Y/Y_n > 0.008856 \\ L &= 903.3 \cdot (Y/Y_n)^{1/3} && \text{for } Y/Y_n \leq 0.008856 \end{aligned}$$

$$a = 500. \cdot [f(X/X_n) - f(Y/Y_n)]$$

$$b = 200. \cdot [f(Y/Y_n) - f(Z/Z_n)]$$

where

$$\begin{aligned} f(t) &= t^{1/3} - 16 && \text{for } t > 0.008856 \\ f(t) &= 7.787 \cdot t + 16/116 && \text{for } t \leq 0.008856 \end{aligned}$$

Here  $Y_n = 1.0$ ,  $X_n = 0.950455$ ,  $Z_n = 1.088753$  for the D65 white point with the CIE chromaticity coordinates  $x_n = 0.312713$ ,  $y_n = 0.329016$ .

The equations above are given on the assumption that initial B, G, R values are normalized to the range  $[0..1]$ . The computed values of the L component are in the range  $[0..100]$ , a and b component values are in the range  $[-128..127]$ .

These values are quantized and scaled to the 8-bit range of 0 to 255 for `ippiBGRTToLab_8u_C3`:

$$\begin{aligned} L &= L \cdot 255./100. \\ a &= (a + 128.) \\ b &= (a + 128.) \end{aligned}$$

or to the 16-bit range of 0 to 65535 for `ippiBGRTToLab_8u16u_C3R`:

$$\begin{aligned} L &= L \cdot 65535./100. \\ a &= (a + 128.) \cdot 255 \\ b &= (a + 128.) \cdot 255 \end{aligned}$$

## Return Values

`ippStsNoErr`                      Indicates no error. Any other value indicates an error.

---

<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## LabToBGR

*Converts a Lab image to the BGR color model.*

---

### Syntax

```
ippStatus ippilabToBGR_8u_C3R(const Ipp8u* pSrc, int srcStep,
                              Ipp8u* pDst, int dstStep, IppiSize roiSize);
ippStatus ippilabToBGR_16u8u_C3R(const Ipp16u* pSrc, int srcStep,
                                  Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippilabToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [CIE Lab](#) image *pSrc* to the BGR image *pDst* in two steps. First, the conversion is carried out into [CIE XYZ](#) format.

To accomplish it, Lab components are transformed back into their original range. This is done for different data types in the following way.

For 8u data type:

```
L = L * 100. / 255.
a = a - 128.
b = b - 128.
```

For 16u data type:

```
L = L * 100./65535.
a = (a/255. - 128.)
b = (b/255.) - 128.)
```

After that, conversion to XYZ format takes place as follows:

```
Y = Yn * P3.
X = Xn * (P + a/500.)3.
Z = Zn * (P - b/200.)3.
```

where

```
P = (L + 16) / 116.
```

After this intermediate conversion is done, the obtained XYZ image is then converted to the destination BGR format using equations defined for the [ippiXYZToRGB](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## RGBToYCC

*Converts an RGB image  
to the YCC color model.*

---

### Syntax

```
IppStatus ippiRGBToYCC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRGBToYCC` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [gamma-corrected](#)  $R'G'B'$  image *pSrc* to the [PhotoY'C'C'](#) image *pDst* according to the following basic equations:

$$\begin{aligned} Y' &= 0.299 \cdot R' + 0.587 \cdot G' + 0.114 \cdot B' \\ C1' &= -0.299 \cdot R' - 0.587 \cdot G' + 0.886 \cdot B' = B' - Y' \\ C2' &= 0.701 \cdot R' - 0.587 \cdot G' - 0.114 \cdot B' = R' - Y' \end{aligned}$$

The equations above are given on the assumption that  $R'$ ,  $G'$ , and  $B'$  values are normalized to the range  $[0..1]$ . In case of the floating-point data type, the input  $R'G'B'$  values must already be in the range  $[0..1]$ . For integer data types, normalization is done by the conversion function internally.

The computed  $Y'$ ,  $C1'$ ,  $C2'$  values are then quantized and converted to fit in the range  $[0..1]$  as follows:

$$\begin{aligned} Y' &= 1. / 1.402 \cdot Y' \\ C1' &= 111.4 / 255. \cdot C1' + 156. / 255. \\ C2' &= 135.64 / 255. \cdot C2' + 137. / 255. \end{aligned}$$

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see [Table 2-2](#) in Chapter 2).

## Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippiStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.



## YCCToRGB

*Converts a YCC image to the RGB color model.*

---

### Syntax

```
IppStatus ippiYCCToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiYCCToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [PhotoY'C'C'](#) image *pSrc* to the R'B'G' image *pDst*. The function `ippiYCCToRGB` first restores normal luminance and chrominance data as:

$$\begin{aligned} Y' &= 1.3584 * Y' \\ C1' &= 2.2179 * (C1' - 156./255.) \\ C2' &= 1.8215 * (C2' - 137./255.) \end{aligned}$$

The equations above are given on the assumption that source *Y*, *C1*, and *C2* values are normalized to the range  $[0..1]$ . In case of the floating-point data type, the input YCC values must already be in the range  $[0..1]$ . For integer data types, normalization is done by the conversion function internally.

After that, YCC data are transformed into RGB format according to the following basic equations:

$$R' = Y' + C2'$$

$$G' = Y' - 0.194 * C1' - 0.509 * C2'$$

$$B' = Y' + C1'$$

In case of integer function flavors, the computed  $R'B'G'$  values are then scaled to the full range of the destination data type (see [Table 2-2](#) in Chapter 2).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## RGBToHLS

*Converts an RGB image to the HLS color model.*

---

### Syntax

```
IppStatus ippRGBToHLS_<mod>(const Ipp<datatype>* pSrc, int srcStep,
                             Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRGBToHLS` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the R'B'G' image `pSrc` to the [HLS](#) image `pDst`. For function flavors operating on the floating point data, source RGB values must be in the range `[0..1]`.

The conversion algorithm from RGB to HLS can be represented in pseudocode as follows:

```
// Lightness:
M1 = max(R,G,B); M2 = max(R,G,B); L = (M1+M2)/2
// Saturation:
if M1 = M2 then // achromatics case
    S = 0
    H = 0
else // chromatics case
    if L <= 0.5 then
        S = (M1-M2) / (M1+M2)
    else
        S = (M1-M2) / (2-M1-M2)
// Hue:
Cr = (M1-R) / (M1-M2)
Cg = (M1-G) / (M1-M2)
Cb = (M1-B) / (M1-M2)
if R = M2 then H = Cb - Cg
if G = M2 then H = 2 + Cr - Cb
if B = M2 then H = 4 + Cg - Cr
H = 60*H
if H < 0 then H = H + 360
```

For floating point function flavors, the computed `H`, `L`, `S` values are scaled to the range `[0..1]`. In case of integer function flavors, these values are scaled to the full range of the destination data type (see [Table 2-2](#) in Chapter 2).

## Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippiStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

## HLSToRGB

*Converts an HLS image to the RGB color model.*

### Syntax

```
IppStatus ippHLSToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippHLSToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [HLS](#) image *pSrc* to the 'R'B'G' image *pDst*. For function flavors operating on the floating point data, source HLS values must be in the range  $[0..1]$ .

The conversion algorithm from HLS to RGB can be represented in pseudocode as follows:

```
if L <= 0.5 then
    M2 = L * (1 + S)
else
    M2 = L + S - L * S
M1 = 2 * L - M2
if S = 0 then
    R = G = B = L
else
    h = H + 120
    if h > 360 then
```

```

        h = h - 360
    if h < 60 then
        R = ( M1 + ( M2 - M1 ) * h / 60)
    else if h < 180 then
        R = M2
    else if h < 240 then
        R = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
    else
        R = M1
    h = H
    if h < 60 then
        G = ( M1 + ( M2 - M1 ) * h / 60)
    else if h < 180 then
        G = M2
    else if h < 240 then
        G = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
    else
        G = M1
    h = H - 120
    if h < 0 then
        h += 360
    if h < 60 then
        B = ( M1 + ( M2 - M1 ) * h / 60)
    else if h < 180 then
        B = M2
    else if h < 240 then
        B = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
    else
        B = M1

```

For floating point function flavors, the computed  $R'$ ,  $G'$ ,  $B'$  values are scaled to the range  $[0..1]$ . In case of integer function flavors, these values are scaled to the full range of the destination data type (see [Table 2-2](#) in Chapter 2).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

## BGRTToHLS

*Converts a BGR image to the HLS color model.*

### Syntax

```
IppStatus ippiBGRTToHLS_8u_AC4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRTToHLS_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiBGRTToHLS_8u_AC4P4R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiBGRTToHLS_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRTToHLS_8u_AP4C4R(const Ipp8u* const pSrc[4],
    int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRTToHLS_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
    Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiBGRTToHLS_8u_AP4R(const Ipp8u* const pSrc[4], int srcStep,
    Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. An array of pointers to ROI in each plane in the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. An array of pointers to ROI in each plane in the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiBGRTToHLS` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the B'G'R' image *pSrc* to the [HLS](#) image *pDst* according to the same [formula](#) as the function `ippiRGBToHLS` does.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## HLSToBGR

*Converts an HLS image to the RGB color model.*

---

### Syntax

```

IppStatus ippHLSToBGR_8u_C3P3R(const Ipp8u* pSrc, int srcStep,
                                Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippHLSToBGR_8u_AC4P4R(const Ipp8u* pSrc, int srcStep,
                                Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippHLSToBGR_8u_AP4R(const Ipp8u* const pSrc[4], int srcStep,
                               Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippHLSToBGR_8u_P3R(const Ipp8u* const pSrc[3], int srcStep,
                              Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippHLSToBGR_8u_AP4C4R(const Ipp8u* const pSrc[4],
                                int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippHLSToBGR_8u_P3C3R(const Ipp8u* const pSrc[3], int srcStep,
                                Ipp8u* pDst, int dstStep, IppiSize roiSize);

```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. An array of pointers to ROI in each plane in the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. An array of pointers to ROI in each plane in the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiHLSToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [HLS](#) image `pSrc` to the B'G'R' image `pDst` according to the same [formula](#) as the function `ippiHLSToRGB` does.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## RGBToHSV

*Converts an RGB image to the HSV color model.*

---

## Syntax

```
IppStatus ippiRGBToHSV_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

<code>8u_C3R</code>	<code>16u_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.



## Description

The function `ippiRGBToHSV` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the R'G'B' image *pSrc* to the [HSV](#) image *pDst*.

The conversion algorithm from RGB to HSV can be represented in pseudocode as follows:

```
// Value:
V = max(R,G,B);
// Saturation:
temp = min(R,G,B);
if V = 0 then // achromatics case
    S = 0//          H = 0
else // chromatics case
    S = (V - temp)/V
// Hue:
Cr = (V - R) / (V - temp)
Cg = (V - G) / (V - temp)
Cb = (V - B) / (V - temp)
if R = V then H = Cb - Cg
if G = V then H = 2 + Cr - Cb
if B = V then H = 4 + Cg - Cr
H = 60*H
if H < 0 then H = H + 360
```

The computed *H*, *S*, *V* values are scaled to the full range of the destination data type (see [Table 2-2](#) in Chapter 2).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## HSVToRGB

*Converts an HSV image to the RGB color model.*

### Syntax

```
IppStatus ippHSVToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep,
                             Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiHSVToRGB` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the R'G'B' image *pSrc* to the [HSV](#) image *pDst*.

The conversion algorithm from HSV to RGB can be represented in pseudocode as follows:

```
if S = 0 then
    R = G = B = V
else
    if H = 360 then
        H = 0
    else
        H = H/60
        I = floor(H)
        F = H - I;
        M = V * ( 1 - S );
        N = V * ( 1 - S * F );
```

```

K = V * ( 1 - S * (1 - F));
if(I == 0)then{ R = V;G = K;B = M;}
if(I == 1)then{ R = N;G = V;B = M;}
if(I == 2)then{ R = M;G = V;B = K;}
if(I == 3)then{ R = M;G = N;B = V;}
if(I == 4)then{ R = K;G = M;B = V;}
if(I == 5)then{ R = V;G = M;B = N;}

```

The computed  $R'$ ,  $G'$ ,  $B'$  values are scaled to the full range of the destination data type (see [Table 2-2](#) in Chapter 2).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## BGRToYCoCg

*Converts a 24-bit BGR image to the YCoCg color model.*

---

### Syntax

```

IppStatus ippIBGRToYCoCg_8u16s_C3P3R(const Ipp8u* pBGR, int bgrStep,
    Ipp16s* pYCC [3], int yccStep, IppiSize roiSize);
IppStatus ippIBGRToYCoCg_8u16s_C4P3R(const Ipp8u* pBGR, int bgrStep,
    Ipp16s* pYCC [3], int yccStep, IppiSize roiSize);

```

### Parameters

<code>pBGR</code>	Pointer to the source image ROI.
<code>bgrStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pYCC</code>	Array of pointers to the destination image ROI in each plane.
<code>yccStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRToYCoCg` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 24-bit BGR image *pSrc* to the [YCoCg](#) image *pDst* according to the following formulas:

$$\begin{aligned} Y &= ((R + 2 * G + B) + 2) / 4 \\ Co &= ((R - B) + 1) / 2 \\ Cg &= ((-R + 2 * G - B) + 2) / 4 \end{aligned}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

---

## SBGRToYCoCg

*Converts a 48-bit BGR image to the YCoCg color model.*

---

## Syntax

```
IppStatus ippiSBGRToYCoCg_<mod>(const Ipp16s* pBGR, int bgrStep,  
    Ipp<dstDatatype>* pYCC[3], int yccStep, IppiSize roiSize);
```

Supported values for *mod*:

<code>16s_C3P3R</code>	<code>16s32s_C3P3R</code>
<code>16s_C4P3R</code>	<code>16s32s_C4P3R</code>

## Parameters

<i>pBGR</i>	Pointer to the source image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pYCC</i>	Array of pointers to the destination image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRToYCoCg` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 48-bit BGR image *pBGR* to the [YCoCg](#) image *pYCC* according to the following formulas:

$$\begin{aligned} Y &= ((R + 2 \cdot G + B) + 2) / 4 \\ Co &= ((R - B) + 1) / 2 \\ Cg &= ((-R + 2 \cdot G - B) + 2) / 4 \end{aligned}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

---

## YCoCgToBGR

*Converts a YCoCg image to the 24-bit BGR image.*

---

## Syntax

```
ippStatus ippiYCoCgToBGR_16s8u_P3C3R(const Ipp16s* pYCC[3], int yccStep,
    Ipp8u* pBGR, int bgrStep, IppiSize roiSize);
ippStatus ippiYCoCgToBGR_16s8u_P3C4R(const Ipp16s* pYCC[3], int yccStep,
    Ipp8u* pBGR, int bgrStep, IppiSize roiSize, Ipp8u aval);
```

## Parameters

<i>pYCC</i>	Array of pointers to the source image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pBGR</i>	Pointer to the destination image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

The function `ippiYCoCrToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the YCoCg image  $pYCC$  to the 24-bit BGR image  $pBGR$  according to the following formulas:

$$R = Y + Co - Cg$$

$$G = Y + Cg$$

$$B = Y - Co - Cg$$

The fourth channel is created by setting channel values to the constant value `aval`.

## Return Values

`ippStsNoErr` Indicates no error. Any other value indicates an error.

`ippStsNullPtrErr` Indicates an error condition if one of the specified pointers is NULL.

---

## YCoCgToSBGR

*Converts a YCoCg image to the 48-bit BGR image.*

---

### Syntax

#### Case 1: Conversion to 3-channel image.

```
ippiYCoCgToSBGR_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp16s*
    pBGR, int bgrStep, IppiSize roiSize);
```

```
IppStatus ippiYCoCgToSBGR_32s16s_P3C3R(const Ipp32s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
```

#### Case 2: Conversion to 4-channel image.

```
IppStatus ippiYCoCgToSBGR_16s_P3C4R(const Ipp16s* pYCC[3], int yccStep,
    Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

```
IppStatus ippiYCoCgToSBGR_32s16s_P3C4R(const Ipp32s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

## Parameters

<i>pYCC</i>	Array of pointers to the source image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pBGR</i>	Pointer to the destination image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

The function `ippiYCoCrToBGR` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [YCoCg](#) image *pYCC* to the 48-bit BGR image *pBGR* according to the following formulas:

$$\begin{aligned} R &= Y + Co - Cg \\ G &= Y + Cg \\ B &= Y - Co - Cg \end{aligned}$$

The fourth channel is created by setting channel values to the constant value *aval*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .

---

## BGRToYCoCg\_Rev

*Converts a 24-bit BGR image to the YCoCg-R color model.*

---

## Syntax

```

IppStatus ippiBGRToYCoCg_Rev_8u16s_C3P3R(const Ipp8u* pBGR, int bgrStep,
    Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);

IppStatus ippiBGRToYCoCg_Rev_8u16s_C4P3R(const Ipp8u* pBGR, int bgrStep,
    Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);

```

## Parameters

<i>pBGR</i>	Pointer to the source image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>yYCC</i>	Array of pointers to the destination image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRTToYCoCg_rev` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 24-bit BGR image *pSrc* to the [YCoCg-R](#) image *pDst* according to the following formulas:

$$\begin{aligned} Co &= R - B \\ t &= B + (Co \gg 1) \\ Cg &= G - t \\ Y &= t + (Cg \gg 1) \end{aligned}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

---

## SBGRTToYCoCg\_Rev

*Converts a 48-bit BGR image to the YCoCg-R color model.*

---

## Syntax

```
ippStatus ippiSBGRTToYCoCg_Rev_<mod>(const Ipp16s* pBGR, int bgrStep,
    Ipp<dstDatatype>* pYCC[3], int yccStep, IppiSize roiSize);
```

Supported values for *mod*:

16s_C3P3R	16s32s_C3P3R
16s_C4P3R	16s32s_C4P3R



## Parameters

<i>pBGR</i>	Pointer to the source image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pYCC</i>	Array of pointers to the destination image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiBGRToYCoCg_Rev` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 48-bit BGR image *pBGR* to the [YCoCg-R](#) image *pYCC* according to the following formulas:

$$\begin{aligned} Co &= R - B \\ t &= B + (Co \gg 1) \\ Cg &= G - t \\ Y &= t + (Cg \gg 1) \end{aligned}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

---

## YCoCgToBGR\_Rev

*Converts a YCoCg-R image to the 24-bit BGR image.*

---

### Syntax

```
ippStatus ippiYCoCgToBGR_Rev_16s8u_P3C3R(const Ipp16s* pYCC[3], int
    yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize);

ippStatus ippiYCoCgToBGR_Rev_16s8u_P3C4R(const Ipp16s* pYCC[3], int
    yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize, Ipp8u aval);
```

## Parameters

<i>pYCC</i>	Array of pointers to the source image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pBGR</i>	Pointer to the destination image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

## Description

The function `ippiYCoCrToBGR_Rev` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [YCoCg-R](#) image *pYCC* to the 24-bit BGR image *pBGR* according to the following formulas:

```
t = Y - (Cg >> 1)
G = Cg + t
B = t - (Co >> 1)
R = B + Co
```

The fourth channel is created by setting channel values to the constant value *aval*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

---

## YCoCgToSBGR\_Rev

*Converts a YCoCg-R image to the 48-bit BGR image.*

---

### Syntax

#### Case 1: Conversion to 3-channel image.

```
ippiYCoCgToSBGR_Rev_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep,
    Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
```

```

IppStatus ippiYCoCgToSBGR_Rev_32s16s_P3C3R(const Ipp32s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);

```

### Case 2: Conversion to 4-channel image.

```

IppStatus ippiYCoCgToSBGR_Rev_16s_P3C4R(const Ipp16s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
IppStatus ippiYCoCgToSBGR_Rev_32s16s_P3C4R(const Ipp32s* pYCC[3], int
    yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);

```

### Parameters

<i>pYCC</i>	Array of pointers to the source image ROI in each plane.
<i>yccStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pBGR</i>	Pointer to the destination image ROI.
<i>bgrStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>aval</i>	Constant value to create the fourth channel.

### Description

The function `ippiYCoCrToBGR_Rev` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the [YCoCg-R](#) image *pYCC* to the 48-bit BGR image *pBGR* according to the following formulas:

$$\begin{aligned}
 t &= Y - (Cg \gg 1) \\
 G &= Cg + t \\
 B &= t - (Co \gg 1) \\
 R &= B + Co
 \end{aligned}$$

The fourth channel is created by setting channel values to the constant value *aval*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

## Color to Gray Scale Conversion

### RGBToGray

*Converts an RGB image to gray scale using fixed transform coefficients.*

#### Syntax

```
IppStatus ippiRGBToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3C1R    16u_C3C1R    16s_C3C1R    32f_C3C1R
8u_AC4C1R    16u_AC4C1R    16s_AC4C1R    32f_AC4C1R
```

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

#### Description

The function `ippiRGBToGray` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function uses the following basic equation to compute luma from nonlinear gamma-corrected red, green, and blue values:

$$Y' = 0.299 * R' + 0.587 * G' + 0.114 * B'$$

Note that the transform coefficients conform to the standard for the NTSC red, green, and blue CRT phosphors.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
--------------------------	---

<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## ColorToGray

*Converts an RGB image to gray scale using custom transform coefficients.*

---

### Syntax

```
IppStatus ippIColorToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp32f
    coeffs[3]);
```

Supported values for *mod*:

```
8u_C3C1R    16u_C3C1R    16s_C3C1R    32f_C3C1R
8u_AC4C1R    16u_AC4C1R    16s_AC4C1R    32f_AC4C1R
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>coeffs</i>	Transform coefficients.

### Description

The function `ippiColorToGray` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function uses the following equation to convert an RGB image to gray scale:

$$Y = coeffs[0] * R + coeffs[1] * G + coeffs[2] * B$$

where the *coeffs* array contains user-defined transform coefficients which must be non-negative and satisfy the condition

$$coeffs[0] + coeffs[1] + coeffs[2] \leq 1$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## Lookup Table Conversion

### LUT

*Maps an image by applying intensity transformation.*

#### Syntax

##### Case 1: Not-in-place operation on one-channel integer data.

```
IppStatus ippILUT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

Supported values for *mod*:

`8u_C1R`      `16s_C1R`

##### Case 2: Not-in-place operation on multi-channel integer data.

```
IppStatus ippILUT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int nLevels[3]);
```

Supported values for *mod*:

`8u_C3R`      `16s_C3R`  
`8u_AC4R`      `16s_AC4R`

```
IppStatus ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int nLevels[4]);
```

Supported values for *mod*:

```
8u_C4R      16s_C4R
```

### Case 3: Not-in-place operation on one-channel floating-point data.

```
IppStatus ippiLUT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, const Ipp32f* pValues,
    const Ipp32f* pLevels, int nLevels);
```

### Case 4: Not-in-place operation on multi-channel floating-point data.

```
IppStatus ippiLUT_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, const Ipp32f* pValues[3],
    const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for *mod*:

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiLUT_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, const Ipp32f* pValues[4],
    const Ipp32f* pLevels[4], int nLevels[4]);
```

### Case 5: In-place operation on one-channel integer data.

```
IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int
    nLevels);
```

Supported values for *mod*:

```
8u_C1IR      16s_C1IR
```

### Case 6: In-place operation on multi-channel integer data.

```
IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
    roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int
    nLevels[3]);
```

Supported values for *mod*:

```
8u_C3IR      16s_C3IR
8u_AC4IR      16s_AC4IR
```

```
IppStatus ippILUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int nLevels[4]);
```

Supported values for *mod*:

```
8u_C4IR      16s_C4IR
```

#### Case 7: In-place operation on one-channel floating-point data.

```
IppStatus ippILUT_32f_C1R(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp32f* pValues, const Ipp32f* pLevels, int nLevels);
```

#### Case 8: In-place operation on multi-channel floating-point data.

```
IppStatus ippILUT_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for *mod*:

```
32f_C3IR
32f_AC4IR
```

```
IppStatus ippILUT_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp32f* pValues[4], const Ipp32f* pLevels[4], int nLevels[4]);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pValues</i>	Pointer to the array of intensity values. In case of multi-channel data, <i>pValues</i> is an array of pointers to the intensity values array for each channel.
<i>pLevels</i>	Pointer to the array of level values. In case of multi-channel data, <i>pLevels</i> is an array of pointers to the level values array for each channel.



*nLevels*                      Number of levels, separate for each channel.

## Description

The function `ippiLUT` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs intensity transformation of the source image *pSrc* using the lookup table (LUT) specified by the arrays *pLevels* and *pValues*. Every source pixel *pSrc*(*x*, *y*) from the range [*pLevels*[*k*], *pLevels*[*k*+1]) is mapped to the destination pixel *pDst*(*x*, *y*) whose value is equal to the intensity *pValues*[*k*].

Length of the *pLevels* and *pValues* arrays is defined by the *nLevels* parameter. Number of level and intensity values is less than *nLevels* by one. Pixels in the *pSrc* image that are not in the range [*pLevels*[0], *pLevels*[*nLevels*-1]) are copied to the *pDst* image without any transformation.

[Figure 6-17](#) shows particular curves that are used in all `ippiLUT` function flavors for mapping. The level values are 0, 64, 128, 192, 256; the intensity values are 20, 60, 160, 180, 230.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error when there is not enough memory for inner buffers.
<code>ippStsLUTNoLevelsErr</code>	Indicates an error when the <i>nLevels</i> is less than 2.

## LUT\_Linear

*Maps an image by applying intensity transformation with linear interpolation.*

### Syntax

#### Case 1: Not-in-place operation on one-channel integer data.

```
IppStatus ippiLUT_Linear_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

Supported values for *mod* :

8u\_C1R          16s\_C1R

#### Case 2: Not-in-place operation on multi-channel integer data.

```
IppStatus ippiLUT_Linear_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int nLevels[3]);
```

Supported values for *mod* :

8u\_C3R          16s\_C3R  
8u\_AC4R          16s\_AC4R

```
IppStatus ippiLUT_Linear_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int nLevels[4]);
```

Supported values for *mod* :

8u\_C4R          16s\_C4R

#### Case 3: Not-in-place operation on one-channel floating-point data.

```
IppStatus ippiLUT_Linear_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues,
    const Ipp32f* pLevels, int nLevels);
```

#### Case 4: Not-in-place operation on multi-channel floating-point data.

```
IppStatus ippiLUT_Linear_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize,
    const Ipp32f* pValues[3], const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for *mod* :

32f\_C3R  
32f\_AC4R

```
IppStatus ippiLUT_Linear_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize,
    const Ipp32f* pValues[4], const Ipp32f* pLevels[4], int nLevels[4]);
```

**Case 5: In-place operation on one-channel integer data.**

```
IppStatus ippiLUT_Linear_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int
    nLevels);
```

Supported values for *mod* :

8u\_C1IR      16s\_C1IR

**Case 6: In-place operation on multi-channel integer data.**

```
IppStatus ippiLUT_Linear_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
```

Supported values for *mod* :

8u\_C3IR      16s\_C3IR  
8u\_AC4IR      16s\_AC4IR

```
IppStatus ippiLUT_Linear_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4],
    int nLevels[4]);
```

Supported values for *mod* :

8u\_C4IR      16s\_C4IR

**Case 7: In-place operation on one-channel floating-point data.**

```
IppStatus ippiLUT_Linear_32f_C1R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues, const Ipp32f* pLevels, int
    nLevels);
```

**Case 8: In-place operation on multi-channel floating-point data.**

```
IppStatus ippiLUT_Linear_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3],
    int nLevels[3]);
```

Supported values for *mod* :

```
32f_C3IR
32f_AC4IR
```

```
IppStatus ippiLUT_Linear_32f_C4R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues[4], const Ipp32f* pLevels[4],
    int nLevels[4]);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pValues</i>	Pointer to the array of intensity values. In case of multi-channel data, <i>pValues</i> is an array of pointers to the intensity values array for each channel.
<i>pLevels</i>	Pointer to the array of level values. In case of multi-channel data, <i>pLevels</i> is an array of pointers to the level values array for each channel.
<i>nLevels</i>	Number of levels, separate for each channel.

## Description

The function `ippiLUT_Linear` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs intensity transformation of the source image *pSrc* using the lookup table (LUT) with linear interpolation between two neighbor levels. LUT is specified by the arrays *pLevels* and *pValues*. Every source pixel *pSrc*(*x*, *y*) from the range [*pLevels*[*k*], *pLevels*[*k*+1]) is mapped to the destination pixel *pDst*(*x*, *y*) whose value is computed according to the following relation:

$$pDst(x, y) = pValues[k] + (pSrc(x, y) - pLevels[k]) \cdot \frac{pValues[k+1] - pValues[k]}{pLevels[k+1] - pLevels[k]}$$

Length of the *pLevels* and *pValues* arrays is defined by the *nLevels* parameter. Pixels in *pSrc* image that are not in the range  $[pLevels[0], pLevels[nLevels-1])$  are copied to *pDst* image without any transformation.

[Figure 6-17](#) shows particular curves that are used in all *ippiLUT* function flavors for mapping. The level values are 0, 64, 128, 192, 256; the intensity values are 20, 60, 160, 180, 230.

### Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error when any of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<i>ippStsMemAllocErr</i>	Indicates an error when there is not enough memory for inner buffers.
<i>ippStsLUTNoLevelsErr</i>	Indicates an error when the <i>nLevels</i> is less than 2.

---

## LUT\_Cubic

*Maps an image by applying intensity transformation with cubic interpolation.*

---

### Syntax

#### Case 1: Not-in-place operation on one-channel integer data.

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues, const Ipp32s* pLevels, int nLevels);
```

Supported values for *mod* :

```
8u_C1R      16s_C1R
```

**Case 2: Not-in-place operation on multi-channel integer data.**

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[3], const Ipp32s* pLevels[3], int nLevels[3]);
```

Supported values for *mod* :

```
8u_C3R      16s_C3R
8u_AC4R     16s_AC4R
```

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32s* pValues[4], const Ipp32s* pLevels[4], int nLevels[4]);
```

Supported values for *mod* :

```
8u_C4R      16s_C4R
```

**Case 3: Not-in-place operation on one-channel floating-point data.**

```
IppStatus ippiLUT_Cubic_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues,
    const Ipp32f* pLevels, int nLevels);
```

**Case 4: Not-in-place operation on multi-channel floating-point data.**

```
IppStatus ippiLUT_Cubic_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues[3],
    const Ipp32f* pLevels[3], int nLevels[3]);
```

Supported values for *mod* :

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiLUT_Cubic_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f* pValues[4],
    const Ipp32f* pLevels[4], int nLevels[4]);
```

**Case 5: In-place operation on one-channel integer data.**

```
IppStatus ippiLUT_Cubic_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues, const Ipp32s* pLevels, int
    nLevels);
```

Supported values for *mod* :

```
8u_C1IR      16s_C1IR
```

### Case 6: In-place operation on multi-channel integer data.

```
IppStatus ippiLUT_Cubic_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
```

Supported values for *mod* :

```
8u_C3IR      16s_C3IR
8u_AC4IR     16s_AC4IR
```

```
IppStatus ippiLUT_Cubic_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32s* pValues[4], const Ipp32s* pLevels[4],
    int nLevels[4]);
```

Supported values for *mod* :

```
8u_C4IR      16s_C4IR
```

### Case 7: In-place operation on one-channel floating-point data.

```
IppStatus ippiLUT_Cubic_32f_C1R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues, const Ipp32f* pLevels, int
    nLevels);
```

### Case 8: In-place operation on multi-channel floating-point data.

```
IppStatus ippiLUT_Cubic_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues[3], const Ipp32f* pLevels[3],
    int nLevels[3]);
```

Supported values for *mod* :

```
32f_C3IR
32f_AC4IR
```

```
IppStatus ippiLUT_Cubic_32f_C4R(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f* pValues[4], const Ipp32f* pLevels[4],
    int nLevels[4]);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pValues</i>	Pointer to the array of intensity values. In case of multi-channel data, <i>pValues</i> is an array of pointers to the intensity values array for each channel.
<i>pLevels</i>	Pointer to the array of level values. In case of multi-channel data, <i>pLevels</i> is an array of pointers to the level values array for each channel.
<i>nLevels</i>	Number of levels, separate for each channel.

## Description

The function `ippiLUT_Cubic` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs intensity transformation of the source image *pSrc* using the lookup table (LUT) with cubic interpolation between neighbor levels. LUT is specified by the arrays *pLevels* and *pValues*.

Every source pixel  $pSrc(x, y)$  from the range  $[pLevels[k], pLevels[k+1])$  is mapped to the destination pixel  $pDst(x, y)$  whose value is computed as

$$pDst(x, y) = A * pSrc(x, y)^3 + B * pSrc(x, y)^2 + C * pSrc(x, y) + D$$

The function operates on the assumption that the cubic polynomial curve passes through the following four points:

(*pLevels*[*k*-1], *pValues*[*k*-1]),  
 (*pLevels*[*k*], *pValues*[*k*]),  
 (*pLevels*[*k*+1], *pValues*[*k*+1]),  
 (*pLevels*[*k*+2], *pValues*[*k*+2])

Based on that, coefficients *A*, *B*, *C*, *D* are computed by solving the following set of linear equations:

$$\begin{aligned} A * pLevels[k-1]^3 + B * pLevels[k-1]^2 + C * pLevels[k-1] + D &= pValues[k-1] \\ A * pLevels[k]^3 + B * pLevels[k]^2 + C * pLevels[k] + D &= pValues[k] \\ A * pLevels[k+1]^3 + B * pLevels[k+1]^2 + C * pLevels[k+1] + D &= pValues[k+1] \\ A * pLevels[k+2]^3 + B * pLevels[k+2]^2 + C * pLevels[k+2] + D &= pValues[k+2] \end{aligned}$$



Length of the *pLevels* and *pValues* arrays is defined by the *nLevels* parameter. Pixels in *pSrc* image that are not in the range  $[pLevels[0], pLevels[nLevels-1])$  are copied to *pDst* image without any transformation.

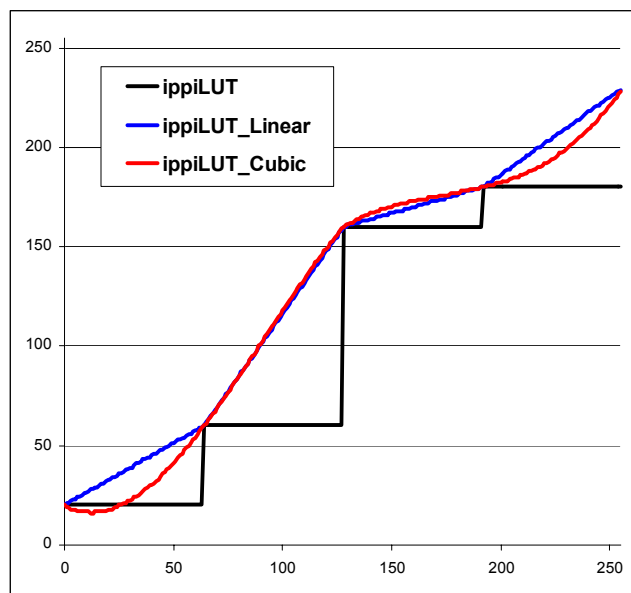
[Figure 6-17](#) shows particular curves that are used in all *ippiLUT* function flavors for mapping. The level values are 0, 64, 128, 192, 256; the intensity values are 20, 60, 160, 180, 230.

### Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippiStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippiStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippiStsMemAllocErr</code>	Indicates an error when there is not enough memory for inner buffers.
<code>ippiStsLUTNoLevelsErr</code>	Indicates an error when the <i>nLevels</i> is less than 2.

**Figure 6-17 Example Mapping Curves Used by the *ippiLUT* Function Flavors**

---



## LUTPalette

*Maps an image by applying intensity transformation in accordance with a palette table.*

### Syntax

```
IppStatus ippiLUTPalette_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, const
    Ipp<dstDatatype>* pTable, int nBitSize);
```

Supported values for *mod*:

```
8u24u_C1R    16u8u_C1R
8u32u_C1R    16u24u_C1R
              16u32u_C1R
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pTable</i>	Pointer to the palette table.
<i>nBitSize</i>	Number of significant bits in the source image.

### Description

The function `ippiLUTPalette` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs intensity transformation of the source image *pSrc* using the palette lookup table *pTable*. This table is a vector with  $2^{nBitSize}$  elements that contain intensity values specified by the user. The function uses *nBitSize* lower bits of intensity value of each source pixel as an index in the *pTable* and assigns the correspondent intensity value from the table to the respective pixel in the destination image *pDst*. The number of significant bits *nBitSize* should be in the range [1, 8] for functions that operate on 8u source images, and [1, 16] for functions that operate on 16u source images.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsOutOfRangeErr</code>	Indicates an error if <i>nBitSize</i> is out of the range.

## Reducing Bit Resolution

---

### ReduceBits

*Reduces the bit resolution of an image.*

---

#### Syntax

##### Case 1: Operation on data of the same source and destination bit depths.

```
IppStatus ippReduceBits_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    int noise, IppiDitherType dtype, int levels);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>16s_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>16s_C3R</code>
<code>8u_C4R</code>	<code>16u_C4R</code>	<code>16s_C4R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>16s_AC4R</code>

##### Case 2: Operation on data of different source and destination bit depths.

```
IppStatus ippReduceBits_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,
    int noise, IppiDitherType dtype, int levels);
```

Supported values for *mod*:

<code>16u8u_C1R</code>	<code>16s8u_C1R</code>	<code>32f8u_C1R</code>	<code>32f16u_C1R</code>	<code>32f16s_C1R</code>
<code>16u8u_C3R</code>	<code>16s8u_C3R</code>	<code>32f8u_C3R</code>	<code>32f16u_C3R</code>	<code>32f16s_C3R</code>

16u8u_C4R	16s8u_C4R	32f8u_C4R	32f16u_C4R	32f16s_C4R
16u8u_AC4R	16s8u_AC4R	32f8u_AC4R	32f16u_AC4R	32f16s_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination ROI in pixels.										
<i>noise</i>	The number specifying the amount of noise added. This parameter is set as a percentage of range $[0..100]$ .										
<i>dtype</i>	The type of dithering to be used. The following types are supported: <table> <tr> <td><code>ippDitherNone</code></td><td>No dithering is done</td></tr> <tr> <td><code>ippDitherStucki</code></td><td>The Stucki's error diffusion dithering algorithm is used</td></tr> <tr> <td><code>ippDitherFS</code></td><td>The Floyd-Steinberg error diffusion dithering algorithm is used</td></tr> <tr> <td><code>ippDitherJUN</code></td><td>The Jarvice-Judice-Ninke error diffusion dithering algorithm is used</td></tr> <tr> <td><code>ippDitherBayer</code></td><td>The Bayer's threshold dithering algorithm is used</td></tr> </table>	<code>ippDitherNone</code>	No dithering is done	<code>ippDitherStucki</code>	The Stucki's error diffusion dithering algorithm is used	<code>ippDitherFS</code>	The Floyd-Steinberg error diffusion dithering algorithm is used	<code>ippDitherJUN</code>	The Jarvice-Judice-Ninke error diffusion dithering algorithm is used	<code>ippDitherBayer</code>	The Bayer's threshold dithering algorithm is used
<code>ippDitherNone</code>	No dithering is done										
<code>ippDitherStucki</code>	The Stucki's error diffusion dithering algorithm is used										
<code>ippDitherFS</code>	The Floyd-Steinberg error diffusion dithering algorithm is used										
<code>ippDitherJUN</code>	The Jarvice-Judice-Ninke error diffusion dithering algorithm is used										
<code>ippDitherBayer</code>	The Bayer's threshold dithering algorithm is used										
<i>levels</i>	The number of output levels for halftoning (dithering); can be varied in the range $[2..(1<depth)]$ , where <i>depth</i> is the bit depth of the destination image.										

## Description

The function `ippiReduceBits` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function reduces the number of intensity levels in each channel of the source image *pSrc* and places the results in respective channels of the destination image *pDst*. Note that for floating point source data type, RGB values must be in the range  $[0..1]$ .

The *levels* parameter sets the resultant number of intensity levels in each channel of the destination image.

If the *noise* value is greater than 0, some random noise is added to the threshold level used in

computations. The amplitude of the noise signal is specified by the *noise* parameter set as a percentage of the destination image luminance range. For the 4x4 ordered dithering mode, the threshold value is determined by the dither matrix used, whereas for the error diffusion dithering mode the input threshold is set as half of the *range* value, where

$range = ((1 \ll depth) - 1) / (levels - 1)$   
and *depth* is the bit depth of the source image.

For floating-point data type,  $range = 1.0 / (levels - 1)$ .

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsNoiseValErr</code>	Indicates an error condition if <i>noise</i> has an illegal value.
<code>ippStsDitherLevelsErr</code>	Indicates an error condition if <i>levels</i> value is out of admissible range.

## Format Conversion

This section describes Intel IPP functions that perform image color conversion without changing the color space. These functions convert pixel-order images to planar format and vice versa, change the number of channels or planes, alter sampling formats and sequences of samples and planes. Several functions additionally perform filtering - deinterlacing and upsampling.

Intel IPP format conversion functions are specified in the YCbCr color space, but as they do not transform color model they may be used to perform described types of conversion for any other color spaces with decoupled luminance and chrominance coordinates (YUV type).

## YCbCr422

Converts 4:2:2 YCbCr image.

### Syntax

```
IppStatus ippiYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the 4:2:2 two-channel source image *pSrc* to the 4:2:2 three-plane image *pDst* and vice versa (see [Table 6-2](#) and [Table 6-3](#) for more details on 4:2:2 planar and pixel-order formats).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> of the first plane is less than 2, or <i>roiSize.height</i> is less than or equal to zero.

## YCbCr422ToYCrCb422

*Converts 4:2:2 YCbCr image to 4:2:2 YCrCb image.*

---

### Syntax

```
IppStatus ippiYCbCr422ToYCrCb422_8u_C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 2.

### Description

The function `ippiYCbCr422ToYCrCb422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:2](#) YCbCr source image *pSrc* to the [4:2:2](#) YCrCb two-channel image *pDst* that has the following sequence of samples: Y0, Cr0, Y1, Cb0, Y2, Cr1, Y3, Cb1, ... (see [Table 6-2](#)). The source image can be either two-channel or three-plane (see [Table 6-2](#) and [Table 6-3](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2.

## YCbCr422ToCbYCr422

*Converts 4:2:2 YCbCr image to 4:2:2 CbYCr image.*

### Syntax

```
IppStatus ippiYCbCr422ToCbYCr422_8u_C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

The functions `ippiYCbCr422ToCbYCr422_8u_C2R` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the order of samples of the [4:2:2](#) two-channel image *pSrc* from Y0, Cb0, Y1, Cr0, Y2, Cb1, Y3, Cr1, ... to Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... (see [Table 6-2](#)) and stores the result in the *pDst*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.



## YCbCr422ToYCbCr420

*Converts YCbCr image from 4:2:2 sampling format to 4:2:0 format.*

---

### Syntax

#### Case 1: Operation on planar data.

```
IppStatus ippiYCbCr422ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
```

#### Case 2: Conversion from pixel-order to planar-order data.

```
IppStatus ippiYCbCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize
    roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.

<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, height and width should be multiple of 2.

## Description

The function `ippiYCbCr422ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function convert the [4:2:2](#) image *pSrc* to the [4:2:0](#) image. The source image can be two-channel or three-plane, destination image always is planar with two or three planes (see [Table 6-2](#) and [Table 6-3](#)). Two-plane image contains luminance samples Y0, Y1, Y2, .. in the first plane *pDstY*, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ... in the second plane *pDstCbCr*.

[Example 6-1](#) shows how to use the function `ippiYCbCr422ToYCbCr420_8u_C2P3R`.

### Example 6-1 Using the Function `ippiYCbCr422ToYCbCr420_8u_C2P3R`

```
{
    Ipp8u*   ImageI420[3];
    int      stepI420[3];
    Ipp8u*   ImageYUY2;
    int      stepYUY2;
    IppiSize roiSize = { 1024, 768 };
    ImageI420[0] = ippiMalloc_8u_C1( roiSize.width, roiSize.height,
    &(stepI420[0]));
    ImageI420[1] = ippiMalloc_8u_C1( roiSize.width, roiSize.height,
    &(stepI420[1]));
    ImageI420[2] = ippiMalloc_8u_C1( roiSize.width, roiSize.height,
    &(stepI420[2]));
    ImageYUY2    = ippiMalloc_8u_C2( roiSize.width, roiSize.height, &stepYUY2 );
    ippiYCbCr422ToYCbCr420_8u_C2P3R( ImageYUY2, stepYUY2, ImageI420, stepI420,
    roiSize );

    ippiFree(ImageI420[0]);
    ippiFree(ImageI420[1]);
    ippiFree(ImageI420[2]);
    ippiFree(ImageYUY2);
}
```

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

---

## YCbCr422ToYCrCb420

*Converts 4:2:2 YCbCr image to 4:2:0 YCrCb image.*

---

### Syntax

```
ippStatus ippYCbCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

### Description

The function `ippYCbCr422ToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function convert the [4:2:2](#) two-channel image *pSrc* that has the following sequence of samples: Y0, Cb0, Y1, Cr0, Y2, Cb1, Y3, Cr1, ... to the [4:2:0](#) three-plane image *pDst* with the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table 6-2](#) and [Table 6-3](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.

`ippStsSizeErr` Indicates an error condition if any field of the *roiSize* is less than 2.

---

## YCbCr422ToYCbCr411

*Converts YCbCr image from 4:2:2 sampling format to 4:1:1 format.*

---

### Syntax

#### Case 1: Operation on planar data.

```
IppStatus ippYCbCr422ToYCbCr411_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippYCbCr422ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
```

#### Case 2: Conversion from pixel-order to planar-order data.

```
IppStatus ippYCbCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippYCbCr422ToYCbCr411_8u_C2P2R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.

<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels.

## Description

The function `ippiYCbCr422ToYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function convert the [4:2:2](#) image *pSrc* to the [4:1:1](#) image. The source image can be two-channel or three-plane (see [Table 6-2](#) for more details), destination image always is planar with two or three planes (see [Table 6-3](#) for more details). Two-plane image contains luminance samples *Y0*, *Y1*, *Y2*, .. in the first plane *pDstY*, and interleaved chrominance samples *Cb0*, *Cr0*, *Cb1*, *Cr1*, ... in the second plane *pDstCbCr*.

The value of the fields of the *roiSize* have certain limitations:

- its width should be multiple of 4 and can not be less than 4 for operation on two-channel images;

- its width should be multiple of 4 and can not be less than 4, and its height should be multiple of 2 and can not be less than 2 for three-plane to two-plane image conversion;

- both height and width should be multiple of 2 and can not be less than 2 for operation on three-plane images.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if corresponding fields of the <i>roiSize</i> is less than specified above values.

## YCrCb422ToYCbCr422

*Converts 4:2:2 YCrCb image to 4:2:2 YCbCr image.*

### Syntax

```
IppStatus ippiYCrCb422ToYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 2.

### Description

The function `ippiYCrCb422ToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:2](#) YCrCb two-channel image *pSrc* (see [Table 6-2](#)) to the [4:2:2](#) YCbCr three-plane image *pDst* (see [Table 6-3](#)).

### Return Values

<code>ppStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2.

## YCrCb422ToYCbCr420

*Converts 4:2:2 YCrCb image to 4:2:0 YCbCr image.*

---

### Syntax

```
IppStatus ippiYCrCb422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, height and width should be multiple of 2.

### Description

The function `ippiYCrCb422ToYCbCr420_8u_C2P3R` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This functions converts [4:2:2](#) YCrCb two-channel image *pSrc* (see [Table 6-2](#)) to the [4:2:0](#) YCbCr three-plane image *pDst* (see [Table 6-3](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

## YCrCb422ToYCbCr411

*Converts 4:2:2 YCrCb image to 4:1:1 YCbCr image.*

### Syntax

```
IppStatus ippiYCrCb422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

### Description

The function `ippiYCrCb422ToYCbCr411_8u_C2P3R` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:2](#) YCrCb two-channel image *pSrc* (see [Table 6-2](#)) to the [4:1:1](#) YCbCr three-plane image *pDst* (see [Table 6-3](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4.



## CbYCr422ToYCbCr422

*Converts 4:2:2 CbYCr image to 4:2:2 YCbCr image.*

---

### Syntax

```
IppStatus ippCbYCr422ToYCbCr422_8u_C2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippCbYCr422ToYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 2.

### Description

The function `ippCbYCr422ToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:2](#) CbYCr two-channel image *pSrc* to the [4:2:2](#) YCbCr two-channel or three-plane image *pDst* (see [Table 6-2](#) and [Table 6-3](#)). The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, .... Two-channel destination image has different sequence of samples: Y0, Cb0, Y1, Cr0, Y2, Cb1, Y3, Cr1, Y4, ....

### Return Values

<code>ppStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2.

## CbYCr422ToYCbCr420

*Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image.*

### Syntax

```
IppStatus ippCbYCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippCbYCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize
    roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

### Description

The function `ippCbYCr422ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:2](#) CbYCr two-channel image *pSrc* to the [4:2:0](#) YCbCr two- or three-plane image *pDst* (see [Table 6-2](#) and [Table 6-3](#)). The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, .... Three-plane destination image has the following order of pointers: Y-plane, Cb-plane, Cr-plane. Two-plane destination image contains luminance samples Y0, Y1, Y2, .. in the first plane *pDstY*, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ... in the second plane *pDstCbCr*.

### Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if any of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

---

## CbYCr422ToYCrCb420

*Converts 4:2:2 CbYCr image to 4:2:0 YCrCb image.*

---

### Syntax

```
IppStatus ippCbYCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

### Description

The function `ippCbYCr422ToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:2](#) CbYCr two-channel image *pSrc* to the [4:2:0](#) YCrCb three-plane image *pDst*. The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, .... The destination image has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table 6-2](#) and [Table 6-3](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

---

## CbYCr422ToYCbCr411

*Converts 4:2:2 CbYCr image to 4:1:1 YCbCr image.*

---

### Syntax

```
IppStatus ippCbYCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

### Description

The function `ippCbYCr422ToYCbCr411_8u_C2P3R` are declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:2](#) CbYCr two-channel image *pSrc* to the [4:1:1](#) YCbCr three-plane image *pDst*. The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, .... The destination image has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see [Table 6-2](#) and [Table 6-3](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4.

---

## YCbCr420

*Converts 4:2:0 YCbCr image.*

---

## Syntax

```

IppStatus ippiYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep,
    IppiSize roiSize);

IppStatus ippiYCbCr420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep,
    const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);

```

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.

<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

### Description

The function `ippiYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:0](#) three-plane (see [Table 6-3](#)) source image *pSrc* to the [4:2:0](#) two-plane image and vice versa. Two-plane image contains luminance samples *Y0*, *Y1*, *Y2*, .. in the first plane, and interleaved chrominance samples *Cb0*, *Cr0*, *Cb1*, *Cr1*, ... in the second plane.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

---

## YCbCr420ToYCbCr422

*Converts YCbCr image from 4:2:0 sampling format to 4:2:2 format.*

---

### Syntax

```
IppStatus ippiYCbCr420ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3],  
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```

IppStatus ippiYCbCr420ToYCbCr422_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);

IppStatus ippiYCbCr420ToYCbCr422_8u_P2C2R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize)

```

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, height and width should be multiple of 2.

## Description

The function `ippiYCbCr420ToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:0](#) planar source image *pSrc* to the [4:2:2](#) image *pDst*. The source image can be two- or three-plane image (see [Table 6-3](#)). The first plane of the two-plane source image *pSrcY* contains luminance samples  $Y_0, Y_1, Y_2, \dots$ , the second plane *pSrcCbCr* contains interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$ . The destination image *pDst* can be three-plane or two-channel (see [Table 6-2](#) and [Table 6-3](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
--------------------------	---

---

<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <code>roiSize</code> is less than 2.

---

## YCbCr420ToYCbCr422\_Filter

*Convert 4:2:0 image to 4:2:2 image with additional filtering.*

---

### Syntax

```
IppStatus ippYCbCr420ToYCbCr422_Filter_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippYCbCr420ToYCbCr422_Filter_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);

IppStatus ippYCbCr420ToYCbCr422_Filter_8u_P2C2R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int
    dstStep, IppiSize roiSize, int layout);
```

### Parameters

<code>pSrc</code>	Array of pointers to the ROI in each plane for a three-plane source image.
<code>srcStep</code>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<code>pSrcY</code>	Pointer to the ROI in the luminance plane for a two-plane source image.
<code>srcYStep</code>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<code>pSrcCbCr</code>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<code>srcCbCrStep</code>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<code>pDst</code>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.



<i>roiSize</i>	Size of the ROI in pixels.								
<i>layout</i>	Slice layout. Possible values: <table> <tr> <td>IPP_UPPER</td><td>Upper (first) slice</td></tr> <tr> <td>IPP_CENTER</td><td>Middle slices</td></tr> <tr> <td>IPP_LOWER</td><td>Lowermost (last) slice</td></tr> <tr> <td>IPP_LOWER &amp;&amp; IPP_UPPER &amp;&amp; IPP_CENTER</td><td>Image is not sliced</td></tr> </table>	IPP_UPPER	Upper (first) slice	IPP_CENTER	Middle slices	IPP_LOWER	Lowermost (last) slice	IPP_LOWER && IPP_UPPER && IPP_CENTER	Image is not sliced
IPP_UPPER	Upper (first) slice								
IPP_CENTER	Middle slices								
IPP_LOWER	Lowermost (last) slice								
IPP_LOWER && IPP_UPPER && IPP_CENTER	Image is not sliced								

## Description

The function `ippiYCbCr420ToYCbCr422_Filter` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:0](#) planar source image *pSrc* to the [4:2:2](#) image *pDst* and performs additional filtering. The source image can be two- or three-plane image (see [Table 6-3](#)). The first plane of the two-plane source image *pSrcY* contains luminance samples *Y0*, *Y1*, *Y2*, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples *Cb0*, *Cr0*, *Cb1*, *Cr1*, .... The destination image *pDst* can be three-plane or two-channel (see [Table 6-2](#) and [Table 6-3](#)).

The function flavors `ippiYCbCr420ToYCbCr422_Filter_8u_P3R` and `ippiYCbCr420ToYCbCr422_Filter_8u_P2P3R` additionally perform the vertical upsampling using a Catmull-Rom interpolation (cubic convolution interpolation). In this case *roiSize.width* should be multiple of 2, and *roiSize.height* should be multiple of 8.

The function `ippiYCbCr420ToYCbCr422_Filter_8u_P2C2R` additionally performs deinterlace filtering. Commonly it is used to process images that are divided into slices. In this case slice *layout* should be specified, since the function processes the first (upper), last (lowermost), and intermediate (middle) slices differently. The height of slices should be a multiple of 16.




---

**CAUTION.** The image slices should be processed exactly in the following order: first slice, intermediate slices, last slice.

---

The function may be applied to a not-sliced image as well. In this case *roiSize.width* and *roiSize.height* should be multiple of 2.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
--------------------------	---

---

<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize.width</code> has wrong value.

---

## YCbCr420ToCbYCr422

*Converts 4:2:0 YCbCr image to 4:2:2 CbYCr image.*

---

### Syntax

```
IppStatus ippiYCbCr420ToCbYCr422_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep,
      const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize
      roiSize);
```

### Parameters

<code>pSrcY</code>	Pointer to the ROI in the luminance plane of the source image.
<code>srcYStep</code>	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
<code>pSrcCbCr</code>	Pointer to the ROI in the interleaved chrominance plane of the source image.
<code>srcCbCrStep</code>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

### Description

The function `ippiYCbCr420ToCbYCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts planar [4:2:0](#) two-plane source image to the pixel-order [4:2:2](#) two-channel image. The first plane of the source image *pSrcY* contains luminance samples Y0, Y1, Y2, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ... . The destination image *pDst* has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ...

### Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if any of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

---

## YCbCr420ToYCrCb420

*Converts 4:2:0 YCbCr image to 4:2:0 YCrCb image.*

---

### Syntax

```
IppStatus ippYCbCr420ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrcY</i>	Pointer to the ROI in the luminance plane of the source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

## Description

The function `ippiYCbCr420ToYCrCb420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:0](#) two-plane source image `pSrc` to the [4:2:0](#) three-plane image `pDst`. The first plane of the source image `pSrcY` contains luminance samples `Y0, Y1, Y2, ...`, the second plane `pSrcCbCr` contains interleaved chrominance samples `Cb0, Cr0, Cb1, Cr1, ...`. The destination image `pDst` has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table 6-3](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <code>roiSize</code> is less than 2.

---

## YCbCr420ToYCrCb420\_Filter

*Convert 4:2:0 YCbCr image to 4:2:0 YCrCb image with deinterlace filtering.*

---

## Syntax

```
IppStatus ippiYCbCr420ToYCrCb420_Filter_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize, int layout);
```

## Parameters

<code>pSrcY</code>	Pointer to the ROI in the luminance plane of the source image.
<code>srcYStep</code>	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
<code>pSrcCbCr</code>	Pointer to the ROI in the interleaved chrominance plane of the source image.
<code>srcCbCrStep</code>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<code>pDst</code>	Array of pointers to the ROI in each plane of the destination image.

<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the destination image planes.								
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.								
<i>layout</i>	Slice layout. Possible values: <table> <tr> <td>IPP_UPPER</td><td>Upper (first) slice</td></tr> <tr> <td>IPP_CENTER</td><td>Middle slices</td></tr> <tr> <td>IPP_LOWER</td><td>Lowermost (last) slice</td></tr> <tr> <td>IPP_LOWER &amp;&amp; IPP_UPPER &amp;&amp; IPP_CENTER</td><td>Image is not sliced</td></tr> </table>	IPP_UPPER	Upper (first) slice	IPP_CENTER	Middle slices	IPP_LOWER	Lowermost (last) slice	IPP_LOWER && IPP_UPPER && IPP_CENTER	Image is not sliced
IPP_UPPER	Upper (first) slice								
IPP_CENTER	Middle slices								
IPP_LOWER	Lowermost (last) slice								
IPP_LOWER && IPP_UPPER && IPP_CENTER	Image is not sliced								

## Description

The functions `ippiYCbCr420ToYCrCb420_Filter` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:0](#) two-plane source image to the [4:2:0](#) three-plane image. The first plane of the source image *pSrcY* contains luminance samples Y0, Y1, Y2, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, .... The destination image *pDst* has the following order of pointers: Y-plane, Cr-plane, Cb-plane.

The function additionally performs deinterlace filtering. Commonly it is used to process sliced images. In this case the slice *layout* should be specified, since the function processes the first (upper), last (lowermost), and intermediate (middle) slices differently. The height of slices should be a multiple of 16. The function may be applied to a not-sliced image as well.




---

**CAUTION.** The image slices should be processed exactly in the following order: first slice, intermediate slices, last slice.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

---

## YCbCr420ToYCbCr411

*Converts YCbCr image from 4:2:0 sampling format to 4:1:1 format.*

---

### Syntax

```
IppStatus ippiYCbCr420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int
    srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int
    dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.

<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

## Description

The function `ippiYCbCr420ToYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:0](#) source image to the [4:1:1](#) destination image. The source two-plane image is converted to destination three-plane image and vice versa. First plane of the two-plane image contains luminance samples  $Y_0, Y_1, Y_2, \dots$ , the second plane contains interleaved chrominance samples  $Cb_0, Cr_0, Cb_1, Cr_1, \dots$ . The three-plane image has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see [Table 6-3](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 2.

---

## YCrCb420ToYCbCr422

*Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image.*

---

## Syntax

```

IppStatus ippiYCrCb420ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiYCrCb420ToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);

```

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
-------------	---

<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, height and width should be multiple of 2.

### Description

The function `ippiYCrCb420ToYCbCr422` is declared in the `ippcc.h` file. This function converts [4:2:0](#) YCrCb three-plane image *pSrc* to the [4:2:2](#) YCbCr three-plane or two-channel image *pDst* (see [Table 6-2](#) and [Table 6-3](#)).

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

---

## YCrCb420ToYCbCr422\_Filter

*Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image with additional filtering.*

---

### Syntax

```
IppStatus ippiYCrCb420ToYCbCr422_Filter_8u_P3R(const Ipp8u* pSrc[3], int  
srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.



<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 2, its height should be multiple of 8.

### Description

The function `ippiYCrCb420ToYCbCr422_Filter` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:0](#) YCrCb three-plane image *pSrc* to the [4:2:2](#) YCbCr three-plane image *pDst* (see [Table 6-3](#)).

Additionally this function performs the vertical upsampling using a Catmull-Rom interpolation (cubic convolution interpolation).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 2 or <i>roiSize.height</i> is less than 8.

---

## YCrCb420ToCbYCr422

*Converts 4:2:0 YCrCb image to 4:2:2 CbYCr image.*

---

### Syntax

```
IppStatus ippiYVToUY420_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

### Description

The function `ippiYCrCb420ToCbYCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:0](#) YCrCb three-plane image *pSrc* (see [Table 6-3](#)) to the [4:2:2](#) CbYCr two-channel image *pDst* with the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... (see [Table 6-2](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

---

## YCrCb420ToYCbCr420

*Converts 4:2:0 YCrCb image to 4:2:0 YCbCr image.*

---

### Syntax

```
IppStatus ippiYCrCb420ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep,
    IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane of a destination image.

<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of a destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

### Description

The function `ippiYCrCb420ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:0](#) YCrCb three-plane image *pSrc* (see [Table 6-3](#)) to the [4:2:0](#) YCbCr two-plane image that contains luminance samples Y0, Y1, Y2, .. in the first plane *pDstY*, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ... in the second plane *pDstCbCr*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any field of the <i>roiSize</i> is less than 2.

---

## YCrCb420ToYCbCr411

*Converts 4:2:0 YCrCb image to 4:1:1 YCbCr image.*

---

### Syntax

```
ippStatus ippiYCrCb420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane of the source image.
-------------	---

<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane of a destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

## Description

The function `ippiYCrCb420ToYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:2:0](#) YCrCb three-plane image *pSrc* (see [Table 6-3](#)) to the [4:1:1](#) YCbCr two-plane image that contains luminance samples Y0, Y1, Y2, .. in the first plane *pDstY*, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ... in the second plane *pDstCbCr*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 2.

---

## YCbCr411

*Converts 4:1:1 YCbCr image.*

---

## Syntax

```
ippStatus ippiYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3],
    Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep,
    IppiSize roiSize);
```

```
IppStatus ippiYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const
    Ipp8u* pSrcCrCb, int srcCrCbStep, Ipp8u* pDst[3],
    int dstStep[3], IppiSize roiSize);
```

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

## Description

The function `ippiYCbCr411` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:1:1](#) three-plane (see [Table 6-3](#)) source image *pSrc* to the [4:1:1](#) two-plane image and vice versa. Two-plane image contains luminance samples Y0, Y1, Y2, .. in the first plane, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ... in the second plane.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4.

---

## YCbCr411ToYCbCr422

*Converts 4:1:1 YCbCr image to 4:2:2 YCbCr image.*

---

### Syntax

```

IppStatus ippYCbCr411ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippYCbCr411ToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippYCbCr411ToYCbCr422_8u_P2P3R(const Ipp8u* pSrcY,
    int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3],
    int dstStep[3], IppiSize roiSize);
IppStatus ippYCbCr411ToYCbCr422_8u_P2C2R(const Ipp8u* pSrcY,
    int srcYStep, const Ipp8u* pSrcCrCb, int srcCrCbStep, Ipp8u* pDst,
    int dstStep, IppiSize roiSize);

```

### Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.

<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

## Description

The function `ippiYCbCr411ToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:1:1](#) planar source image *pSrc* to the [4:2:2](#) image *pDst*. The first plane of the two-plane source image *pSrcY* contains luminance samples *Y0*, *Y1*, *Y2*, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples *Cb0*, *Cr0*, *Cb1*, *Cr1*, .... The destination image *pDst* can be either three-plane (see [Table 6-3](#)) or two-channel image (see [Table 6-2](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4.

---

## YCbCr411ToYCrCb422

*Converts 4:1:1 YCbCr image to 4:2:2 YCrCb image.*

---

## Syntax

```

IppStatus ippiYCbCr411ToYCrCb422_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr411ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);

```

**Parameters**

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4.

**Description**

The function `ippiYCbCr411ToYCbCr422` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:1:1](#) three-plane image *pSrc* to the [4:2:2](#) two-channel or three-plane image *pDst* with different order of components. The source image has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see [Table 6-3](#)). The three-plane destination image has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table 6-3](#)), and two-channel destination image has the following sequence of samples: Y0, Cr0, Y1, Cb0, Y2, Cr1, Y3, Cb1, ... (see [Table 6-2](#)).

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4.

---

**YCbCr411ToYCbCr420**

*Converts 4:1:1 YCbCr image to 4:2:0 YCbCr image.*

---

**Syntax**

```
IppStatus ippiYCbCr411ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```



```

IppStatus ippiYCbCr411ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int
    srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int
    dstCbCrStep, IppiSize roiSize);

IppStatus ippiYCbCr411ToYCbCr420_8u_P2P3R(const Ipp8u* pSrcY,
    int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3],
    int dstStep[3], IppiSize roiSize);

```

## Parameters

<i>pSrc</i>	Array of pointers to the ROI in each plane for a three-plane source image.
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
<i>pSrcY</i>	Pointer to the ROI in the luminance plane for a two-plane source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane for a three-plane destination image.
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
<i>pDstY</i>	Pointer to the ROI in the luminance plane for a two-plane destination image.
<i>dstYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
<i>pDstCbCr</i>	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
<i>dstCbCrStep</i>	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

## Description

The function `ippiYCbCr411ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:1:1](#) planar source image *pSrc* (see [Table 6-3](#)) to the [4:2:0](#) planar image *pDst*. Both source and destination images can be three- or two-plane. Three-plane images has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see [Table 6-3](#)). Two-plane images contain luminance samples Y0, Y1, Y2, .. in the first plane, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ... in the second plane.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 2.

---

## YCbCr411ToYCrCb420

*Converts 4:1:1 YCbCr image to 4:2:0 YCrCb image.*

---

## Syntax

```
IppStatus ippiYCbCr411ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY,
    int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3],
    int dstStep[3], IppiSize roiSize);
```

## Parameters

<i>pSrcY</i>	Pointer to the ROI in the luminance plane of the source image.
<i>srcYStep</i>	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
<i>pSrcCbCr</i>	Pointer to the ROI in the interleaved chrominance plane of the source image.
<i>srcCbCrStep</i>	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
<i>pDst</i>	Array of pointers to the ROI in each plane of the destination image.

<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in each plane of the destination image.
<i>roiSize</i>	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

## Description

The function `ippiYCbCr411ToYCbCr420` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts [4:1:1](#) two-plane source image *pSrc* to the [4:2:0](#) three-plane image *pDst* with different order of components. The first plane of the source image *pSrcY* contains luminance samples *Y0*, *Y1*, *Y2*, ..., the second plane *pSrcCbCr* contains interleaved chrominance samples *Cb0*, *Cr0*, *Cb1*, *Cr1*, ... .The destination image has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see [Table 6-3](#)),

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize.width</i> is less than 4 or <i>roiSize.height</i> is less than 2.

## Color Twist

Color twist conversion functions use values of all color channels of a source pixel to compute the resultant destination channel value. The destination channel value is obtained as the result of multiplying the corresponding row of the color-twist matrix by the vector of source pixel channel values.

For example, if (*r*, *g*, *b*) is a source pixel, then the destination pixel values (*R*, *G*, *B*) are computed as follows:

$$\begin{aligned} R &= t_{11} * r + t_{12} * g + t_{13} * b + t_{14} \\ G &= t_{21} * r + t_{22} * g + t_{23} * b + t_{24} \\ B &= t_{31} * r + t_{32} * g + t_{33} * b + t_{34} \end{aligned}$$

where

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \end{bmatrix}$$

is the color twist matrix.

The color twist matrix used by Intel IPP functions is a matrix of size 3x4, or 4x4 with floating-point elements. The matrix elements are specific for each particular type of color conversion.

---

## ColorTwist

*Applies a color twist matrix to an image with floating-point pixel values.*

---

### Syntax

#### Case 1: Not-in-place operation on pixel-order data.

```
IppStatus ippiColorTwist_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize,
    const Ipp32f twist[3][4]);
```

Supported values for *mod*:

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiColorTwist_32f_C4R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize,
    const Ipp32f twist[3][4]);
```

#### Case 2: Not-in-place operation on planar data.

```
IppStatus ippiColorTwist_32f_P3R(const Ipp32f* const pSrc[3], int srcStep,
    Ipp32f* const pDst[3], int dstStep, IppiSize roiSize,
    const Ipp32f twist[3][4]);
```

### Case 3: In-place operation on pixel-order data.

```
IppStatus ippiColorTwist_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for *mod*:

```
32f_C3IR
32f_AC4IR
```

### Case 4: In-place operation on planar data.

```
IppStatus ippiColorTwist_32f_IP3R(Ipp32f* const pSrcDst[3],
    int srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>twist</i>	The array containing color-twist matrix elements.

## Description

The function `ippiColorTwist` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the color-twist matrix to all three color channels in the source image with floating-point pixel values to obtain the resulting data in the destination image. The destination channel value is obtained as the result of multiplying the corresponding row of the color-twist matrix by the vector of source pixel channel values.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
--------------------------	---

<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## ColorTwist32f

*Applies a color twist matrix to an image with integer pixel values.*

---

### Syntax

#### Case 1: Not-in-place operation on pixel-order data.

```
IppStatus ippIColorTwist32f_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp32f twist[3][4]);
```

Supported values for *mod*:

8u_C3R	8s_C3R	16u_C3R	16s_C3R
8u_AC4R	8s_AC4R	16u_AC4R	16s_AC4R

#### Case 2: Not-in-place operation on planar data.

```
IppStatus ippIColorTwist32f_<mod>(const Ipp<datatype>* const pSrc[3],
    int srcStep, Ipp<datatype>* const pDst[3], int dstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for *mod*:

8u_P3R	8s_P3R	16u_P3R	16s_P3R
--------	--------	---------	---------

#### Case 3: In-place operation on pixel-order data.

```
IppStatus ippIColorTwist32f_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for *mod*:

8u_C3IR	8s_C3IR	16u_C3IR	16s_C3IR
8u_AC4IR	8s_AC4IR	16u_AC4IR	16s_AC4IR

#### Case 4: In-place operation on planar data.

```
IppStatus ippiColorTwist32f_<mod>(Ipp<datatype>* const pSrcDst[3],
    int srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for *mod*:

8u\_IP3R      8s\_IP3R      16u\_IP3R      16s\_IP3R

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>twist</i>	The array containing color-twist matrix elements.

#### Description

The function `ippiColorTwist32f` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the color-twist matrix to all three color channel values in the integer source image to obtain the resulting data in the destination image. For example, the conversion from RGB to YCbCr format can be done as

$$\begin{aligned}
 Y &= 0.299 * R + 0.587 * G + 0.114 * B \\
 Cb &= -0.16874 * R - 0.33126 * G + 0.5 * B + 0.5 \\
 Cr &= 0.5 * R - 0.41869 * G - 0.08131 * B + 0.5
 \end{aligned}$$

which can be described in terms of the following color twist matrix:

```
0.29900f  0.58700f  0.11400f  0.000f
-0.16874f -0.33126f  0.50000f  128.0f
0.50000f  -0.41869f -0.08131f  128.0f
```

Color-twist matrices may also be used to perform many other color conversions.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## Gamma Correction

### GammaFwd

*Performs gamma-correction  
of the source image with RGB data.*

#### Syntax

##### Case 1: Not-in-place operation on integer pixel-order data.

```
IppStatus ippGammaFwd_<mod>(const Ipp<datatype>* pSrc, int srcStep,
                             Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

<code>8u_C3R</code>	<code>16u_C3R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>

##### Case 2: Not-in-place operation on integer planar data.

```
IppStatus ippGammaFwd_<mod>(const Ipp<datatype>* const pSrc[3],
                             int srcStep, Ipp<datatype>* const pDst[3], int dstStep,
                             IppiSize roiSize);
```

Supported values for *mod*:

<code>8u_P3R</code>	<code>16u_P3R</code>
---------------------	----------------------



**Case 3: Not-in-place operation on floating-point pixel-order data.**

```
IppStatus ippiGammaFwd_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for *mod*:

```
32f_C3R
32f_AC4R
```

**Case 4: Not-in-place operation on floating-point planar data.**

```
IppStatus ippiGammaFwd_32f_P3R (const Ipp32f* const pSrc[3], int srcStep,
    Ipp32f* const pDst[3], int dstStep, IppiSize roiSize, Ipp32f vMin,
    Ipp32f vMax);
```

**Case 5: In-place operation on integer pixel-order data.**

```
IppStatus ippiGammaFwd_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3IR      16u_C3IR
8u_AC4IR      16u_AC4IR
```

**Case 6: In-place operation on integer planar data.**

```
IppStatus ippiGammaFwd_<mod>(Ipp<datatype>* const pSrcDst[3],
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_IP3R      16u_IP3R
```

**Case 7: In-place operation on floating-point pixel-order data.**

```
IppStatus ippiGammaFwd_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for *mod*:

```
32f_C3IR
32f_AC4IR
```

**Case 8: In-place operation on floating-point planar data.**

```
IppStatus ippiGammaFwd_32f_IP3R (Ipp32f* const pSrcDst[3],
    int srcDstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

## Parameters

<i>pSrc</i>	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>vMin</i> , <i>vMax</i>	Minimum and maximum values of the input floating-point data.

## Description

The function `ippiGammaFwd` is declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs [gamma-correction](#) of the source image with RGB data. It uses the following basic equations to convert an RGB image to the gamma-corrected R'G'B' image:

```
for R, G, B < 0.018
    R' = 4.5 * R
    G' = 4.5 * G
    B' = 4.5 * B

for R, G, B ≥ 0.018
    R' = 1.099 * R0.45 - 0.099
    G' = 1.099 * G0.45 - 0.099
    B' = 1.099 * B0.45 - 0.099
```

Note that the channel intensity values are normalized to fit in the range of [0..1]. The gamma value is equal to  $1/0.45 = 2.22$  in conformity with [ITU709](#) specification.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsGammaRangeErr</code>	Indicates an error condition if the input data bounds are incorrect, that is <i>vMax</i> is less than or equal to <i>vMin</i> .

---

## Gammaln

*Converts a gamma-corrected R'G'B' image back to the original RGB image.*

---

### Syntax

#### Case 1: Not-in-place operation on integer pixel-order data.

```
IppStatus ippGammaInv_<mod>(const Ipp<datatype>* pSrc, int srcStep,
                             Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
8u_C3R      16u_C3R
8u_AC4R      16u_AC4R
```

#### Case 2: Not-in-place operation on integer planar data.

```
IppStatus ippGammaInv_<mod>(const Ipp<datatype>* const pSrc[3],
                             int srcStep, Ipp<datatype>* const pDst[3], int dstStep,
                             IppiSize roiSize);
```

Supported values for *mod*:

```
8u_P3R      16u_P3R
```

#### Case 3: Not-in-place operation on floating-point pixel-order data.

```
IppStatus ippGammaInv_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
                             int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for *mod*:

32f\_C3R  
32f\_AC4R

#### Case 4: Not-in-place operation on floating-point planar data.

```
IppStatus ippiGammaInv_32f_P3R (const Ipp32f* const pSrc[3], int srcStep,
    Ipp32f* const pDst[3], int dstStep, IppiSize roiSize, Ipp32f vMin,
    Ipp32f vMax);
```

#### Case 5: In-place operation on integer pixel-order data.

```
IppStatus ippiGammaInv_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u\_C3IR      16u\_C3IR  
8u\_AC4IR      16u\_AC4IR

#### Case 6: In-place operation on integer planar data.

```
IppStatus ippiGammaInv_<mod>(Ipp<datatype>* const pSrcDst[3],
    int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_IP3R      16u\_IP3R

#### Case 7: In-place operation on floating-point pixel-order data.

```
IppStatus ippiGammaInv_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for *mod*:

32f\_C3IR  
32f\_AC4IR

#### Case 8: In-place operation on floating-point planar data.

```
IppStatus ippiGammaInv_32f_IP3R (Ipp32f* const pSrcDst[3], int srcDstStep,
    IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

### Parameters

*pSrc*      Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>vMin</i> , <i>vMax</i>	Minimum and maximum values of the input floating-point data.

## Description

The function `ippiGammaInvis` declared in the `ippcc.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a [gamma-corrected](#) R'G'B' image back to the original RGB image. It uses the following equations:

for  $R', G', B' < 0.0812$

$$R = R' / 4.5$$

$$G = G' / 4.5$$

$$B = B' / 4.5$$

for  $R', G', B' \geq 0.0812$

$$r = \left( \frac{R' + 0.099}{1.099} \right)^{2.2}$$

$$g = \left( \frac{G' + 0.099}{1.099} \right)^{2.2}$$

$$b = \left( \frac{B' + 0.099}{1.099} \right)^{2.2}$$

Note that the channel intensity values are normalized to fit in the range of [0..1]. The gamma value is equal to  $1/0.45 = 2.22$  in conformity with [\[ITU709\]](#) specification.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsGammaRangeErr</code>	Indicates an error condition if the input data bounds are incorrect, that is <i>vMax</i> is less than or equal to <i>vMin</i> .

# Threshold and Compare Operations

## 7

This chapter describes the Intel IPP image processing functions that operate on a pixel-by-pixel basis: threshold and compare functions.

[Table 7-1](#) lists all functions of this group:

**Table 7-1 Image Threshold and Compare Functions**

Function Base Name	Operation
<b>Thresholding</b>	
<a href="#">Threshold</a>	Performs thresholding of pixel values in an image, using the specified compare operation
<a href="#">Threshold_GT</a>	Thresholds pixel values of an image, using the comparison for “greater than”
<a href="#">Threshold_LT</a>	Thresholds pixel values of an image, using the comparison for “less than”
<a href="#">Threshold_Val</a>	Thresholds pixel values of an image. Pixels that satisfy the compare condition are set to a specified value
<a href="#">Threshold_GTVal</a>	Thresholds pixel values of an image. Pixels that are greater than threshold are set to a specified value
<a href="#">Threshold_LTVal</a>	Thresholds pixel values of an image. Pixels that are less than threshold are set to a specified value
<a href="#">Threshold_LTValGTVal</a>	Performs double thresholding of pixel values in an image
<b>Comparison</b>	
<a href="#">Compare</a>	Compares pixel values of two images using a specified compare operation
<a href="#">CompareC</a>	Compares pixel values of a source image to a given value using a specified compare operation

**Table 7-1 Image Threshold and Compare Functions (continued)**

Function Base Name	Operation
<a href="#">CompareEqualEps</a>	Compares two images with floating-point data, testing whether pixel values are equal within a certain tolerance <i>eps</i>
<a href="#">CompareEqualEpsC</a>	Tests whether floating-point pixel values of an image are equal to a given value within a certain tolerance <i>eps</i>

## Thresholding

The threshold functions change pixel values depending on whether they are less or greater than the specified *threshold*.

The type of comparison operation used to threshold pixel values is specified by the *ippCmpOp* parameter; this operation can be either “greater than” or “less than” (see [Structures and Enumerators](#) in chapter 2 for more information). For some thresholding functions the type of comparison operation is fixed.

If an input pixel value satisfies the compare condition, the corresponding output pixel is set to the fixed value that is specific for a given threshold function flavor. Otherwise, it is either not changed, or set to another fixed value, which is defined in a particular function description.

For images with multi-channel data, the compare conditions should be set separately for each channel.

## Threshold

*Performs thresholding  
of pixel values in an image buffer.*

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> threshold, IppCmpOp ippCmpOp);
```



Supported values for *mod* :

8u\_C1R          16s\_C1R          32f\_C1R

### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp<datatype> threshold[3], IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u\_C3R          16s\_C3R          32f\_C3R  
8u\_AC4R          16s\_AC4R          32f\_AC4R

### Case 3: In-place operation on one-channel data

```
IppStatus ippThreshold_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp<datatype> threshold, IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u\_C1IR          16s\_C1IR          32f\_C1IR

### Case 4: In-place operation on multi-channel data

```
IppStatus ippThreshold_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp<datatype> threshold[3],
    IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u\_C3IR          16s\_C3IR          32f\_C3IR  
8u\_AC4IR          16s\_AC4IR          32f\_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).

<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.
<i>ippCmpOp</i>	The operation specified for comparing pixel values and the <i>threshold</i> . Comparison for either “less than” or “greater than” can be used.

## Description

The function `ippiThreshold` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value using the *ippCmpOp* comparison operation.

If the result of the compare is true, the corresponding output pixel is set to the *threshold* value. Otherwise, it is set to the source pixel value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Threshold\_GT

*Performs thresholding of pixel values in an image, using the comparison for “greater than”.*

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_GT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> threshold);
```

Supported values for *mod* :

8u\_C1R          16s\_C1R          32f\_C1R

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_GT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp<datatype> threshold[3]);
```

Supported values for *mod* :

8u\_C3R          16s\_C3R          32f\_C3R  
8u\_AC4R          16s\_AC4R          32f\_AC4R

#### Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_GT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp<datatype> threshold);
```

Supported values for *mod* :

8u\_C1IR          16s\_C1IR          32f\_C1IR

#### Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_GT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for *mod* :

8u\_C3IR          16s\_C3IR          32f\_C3IR  
8u\_AC4IR          16s\_AC4IR          32f\_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.

## Description

The function `ippiThreshold_GT` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs thresholding of pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for “greater than”.

If the result of the compare is true, the corresponding output pixel is set to the *threshold* value. Otherwise, it is set to the source pixel value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Threshold\_LT

*Performs thresholding of pixel values in an image buffer, using the comparison for “less than”.*

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_LT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> threshold);
```

Supported values for *mod* :

8u\_C1R          16s\_C1R          32f\_C1R

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LT_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp<datatype> threshold[3]);
```

Supported values for *mod* :

8u\_C3R          16s\_C3R          32f\_C3R  
8u\_AC4R          16s\_AC4R          32f\_AC4R

#### Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_LT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp<datatype> threshold);
```

Supported values for *mod* :

8u\_C1IR          16s\_C1IR          32f\_C1IR

#### Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_LT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for *mod* :

8u\_C3IR          16s\_C3IR          32f\_C3IR  
8u\_AC4IR          16s\_AC4IR          32f\_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.

## Description

The function `ippiThreshold_LT` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs thresholding of pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for “less than”. If the result of the compare is true, the corresponding output pixel is set to the *threshold* value.

Otherwise, it is set to the source pixel value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Threshold\_Val

*Performs thresholding of pixel values in an image buffer. Pixels that satisfy the compare condition are set to a specified value.*

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> threshold, Ipp<datatype> value, IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u\_C1R          16s\_C1R          32f\_C1R

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>
    threshold[3], const Ipp<datatype> value[3], IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u\_C3R          16s\_C3R          32f\_C3R  
8u\_AC4R          16s\_AC4R          32f\_AC4R

#### Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, Ipp<datatype> threshold, Ipp<datatype> value,
    IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u\_C1IR          16s\_C1IR          32f\_C1IR

#### Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype>
    value[3], IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u\_C3IR          16s\_C3IR          32f\_C3IR

8u\_AC4IR      16s\_AC4IR      32f\_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold value to use for each pixel. In case of multi-channel data, an array of 3 threshold values (one for each color channel) is used.
<i>value</i>	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of 3 output values (one for each color channel) is used.
<i>ippCmpOp</i>	The operation to use for comparing pixel values and the <i>threshold</i> . Comparison for either “less than” or “greater than” can be used.

## Description

The function `ippiThreshold_Val` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value using the *ippCmpOp* comparison operation. If the result of the compare is true, the corresponding output pixel is set to the specified *value*. Otherwise, it is set to the source pixel value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.



<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> , <code>dstStep</code> , or <code>srcDstStep</code> has a zero or negative value.

---

## Threshold\_GTVal

*Performs thresholding of pixel values in an image.  
Pixels that are greater than threshold, are set to a specified value.*

---

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> threshold, Ipp<datatype> value);
```

Supported values for `mod` :

`8u_C1R`      `16s_C1R`      `32f_C1R`

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>
    threshold[3], const Ipp<datatype> value[3]);
```

Supported values for `mod` :

`8u_C3R`      `16s_C3R`      `32f_C3R`  
`8u_AC4R`      `16s_AC4R`      `32f_AC4R`

```
IppStatus ippThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>
    threshold[4], const Ipp<datatype> value[4]);
```

Supported values for `mod` :

`8u_C4R`      `16s_C4R`      `32f_C4R`

### Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold,
    Ipp<datatype> value);
```

Supported values for *mod* :

8u\_C1IR      16s\_C1IR      32f\_C1IR

### Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[3],
    const Ipp<datatype> value[3]);
```

Supported values for *mod* :

8u\_C3IR      16s\_C3IR      32f\_C3IR  
8u\_AC4IR      16s\_AC4IR      32f\_AC4IR

```
IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[4],
    const Ipp<datatype> value[4]);
```

Supported values for *mod* :

8u\_C4IR      16s\_C4IR      32f\_C4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold value to use for each pixel. In case of multi-channel data, an array of threshold values (one for each channel) is used.

*value*                      The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of output values (one for each channel) is used.

## Description

The function `ippiThreshold_LTVAl` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for “greater than”.

If the result of the compare is true, the corresponding output pixel is set to the specified *value*. Otherwise, it is set to the source pixel value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

---

## Threshold\_LTVAl

*Performs thresholding of pixel values in an image.  
Pixels that are less than threshold, are set to a specified value.*

---

## Syntax

### Case 1: Not-in-place operation on one-channel data

```
ippStatus ippiThreshold_LTVAl<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> threshold, Ipp<datatype> value);
```

Supported values for *mod* :

`8u_C1R`            `16s_C1R`            `32f_C1R`

### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);
```

Supported values for *mod* :

8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]);
```

Supported values for *mod* :

8u_C4R	16s_C4R	32f_C4R
--------	---------	---------

### Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold,
    Ipp<datatype> value);
```

Supported values for *mod* :

8u_C1IR	16s_C1IR	32f_C1IR
---------	----------	----------

### Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, const Ipp<datatype>
    threshold[3], const Ipp<datatype> value[3]);
```

Supported values for *mod* :

8u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16s_AC4IR	32f_AC4IR

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, const Ipp<datatype>
    threshold[4], const Ipp<datatype> value[4]);
```

Supported values for *mod* :

8u_C4IR	16s_C4IR	32f_C4IR
---------	----------	----------

## Parameters

*pSrc*                                      Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>threshold</i>	The threshold value to use for each pixel. In case of multi-channel data, an array of threshold values (one for each channel) is used.
<i>value</i>	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of output values (one for each channel) is used.

## Description

The function `ippiThreshold_LTVal` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using the specified level *threshold*. Pixel values in the source image are compared to the *threshold* value for “less than”. If the result of the compare is true, the corresponding output pixel is set to the specified *value*. Otherwise, it is set to the source pixel value.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.

## Threshold\_LTValGTVal

*Performs double thresholding  
of pixel values in an image buffer.*

---

### Syntax

#### Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    Ipp<datatype> thresholdLT, Ipp<datatype> valueLT,
    Ipp<datatype> thresholdGT, Ipp<datatype> valueGT);
```

Supported values for *mod* :

8u\_C1R          16s\_C1R          32f\_C1R

#### Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const Ipp<datatype> thresholdLT[3], const Ipp<datatype> valueLT[3],
    const Ipp<datatype> thresholdGT[3], const Ipp<datatype> valueGT[3]);
```

Supported values for *mod* :

8u\_C3R          16s\_C3R          32f\_C3R  
8u\_AC4R          16s\_AC4R          32f\_AC4R

#### Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, Ipp<datatype> thresholdLT,
    Ipp<datatype> valueLT, Ipp<datatype> thresholdGT,
    Ipp<datatype> valueGT);
```

Supported values for *mod* :

8u\_C1IR          16s\_C1IR          32f\_C1IR

#### Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst,
    int srcDstStep, IppiSize roiSize, const Ipp<datatype> thresholdLT[3],
    const Ipp<datatype> valueLT[3], const Ipp<datatype> thresholdGT[3],
    const Ipp<datatype> valueGT[3]);
```

Supported values for *mod* :

8u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16s_AC4IR	32f_AC4IR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI (for the in-place operation).
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>thresholdLT</i>	The lower threshold value to use for each pixel. In case of multi-channel data, an array of 3 lower threshold values (one for each color channel) is used.
<i>valueLT</i>	The lower output value to be set for each pixel that is less than <i>thresholdLT</i> . In case of multi-channel data, an array of 3 lower output values (one for each color channel) is used.
<i>thresholdGT</i>	The upper threshold value to use for each pixel. In case of multi-channel data, an array of 3 upper threshold values (one for each color channel) is used.
<i>valueGT</i>	The upper output value to be set for each pixel that exceeds <i>thresholdGT</i> . In case of multi-channel data, an array of 3 upper output values (one for each color channel) is used.

## Description

The function `ippiThreshold_LTValGTVal` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function thresholds pixels in the source image *pSrc* using two specified levels *thresholdLT* and *thresholdGT*. Pixel values in the source image are compared to these levels. If the pixel value is less than *thresholdLT*, the corresponding output pixel is set to *valueLT*. If the pixel value is greater than *thresholdGT*, the output pixel is set to *valueGT*. Otherwise, it is set to the source pixel value.

The value of *thresholdLT* should be less than or equal to *thresholdGT*.

The code example below illustrates thresholding with two levels:

### Example 7-1 Thresholding an Image

---

```
IppStatus threshold( void ) {
    Ipp8u x[5*4];
    IppiSize roi = {5,4};
    int i;
    for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;
    return ippiThreshold_LTValGTVal_8u_C1IR( x, 5, roi, 2,1,6,7, );
}
```

The destination image *x* contains:

```
01 01 02 03 04
05 06 07 07 07
07 07 07 07 07
07 07 07 07 07
```

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsThresholdErr</code>	Indicates an error when <i>thresholdLT</i> is greater than <i>thresholdGT</i> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.



## Compare Operations

This section describes functions that compare images. Each compare function writes its results to a 1-channel `Ipp8u` output image. The output pixel is set to a non-zero value if the corresponding input pixel(s) satisfy the compare condition; otherwise, the output pixel is set to 0. You can compare either two images, or an image and a constant value, using the following compare conditions: “greater”, “greater or equal”, “less”, “less or equal”, “equal”. Compare condition is specified as a function argument of `IppCmpOp` type (see [Structures and Enumerators](#) in chapter 2 for more information). Images containing floating-point data can also be compared for being equal within a given tolerance *eps*.

For images with multi-channel data, the compare condition for a given pixel is true only when each color channel value of that pixel satisfies this condition.

---

## Compare

*Compares pixel values of two images  
using a specified compare operation.*

---

### Syntax

```
IppStatus ippiCompare_<mod>(const Ipp<datatype>* pSrc1, int src1Step,  
    const Ipp<datatype>* pSrc2, int src2Step, Ipp8u* pDst, int dstStep,  
    IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<i>pSrc1, pSrc2</i>	Pointers to the source image ROIs.
<i>src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>ippCmpOp</i>	Compare operation to be used for comparing the pixel values.

## Description

The function `ippiCompare` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function compares the corresponding pixels of ROI in two source images *pSrc1*, *pSrc2* using the *ippCmpOp* compare operation, and writes the results to a 1-channel `Ipp8u` image *pDst*. If the result of the compare is true, the corresponding output pixel is set to an `IPP_MAX_8U` value; otherwise, it is set to 0.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>src1Step</i> , <i>src2Step</i> , or <i>dstStep</i> has a zero or negative value.

---

## CompareC

*Compares pixel values of a source image to a given value using a specified compare operation.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype> value, Ipp8u* pDst, int dstStep, IppiSize roiSize,
    IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u\_C1R          16s\_C1R          32f\_C1R

## Case 2: Operation on multi-channel data

```
IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[3], Ipp8u* pDst, int dstStep,
    IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u\_C3R          16s\_C3R          32f\_C3R  
8u\_AC4R          16s\_AC4R          32f\_AC4R

```
IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp<datatype> value[4], Ipp8u* pDst, int dstStep,
    IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for *mod* :

8u\_C4R          16s\_C4R          32f\_C4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The value to compare each pixel to. In case of multi-channel data, an array of separate values (one for each channel).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>ippCmpOp</i>	Compare operation to be used for comparing the pixel values.

## Description

The function `ippiCompareC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function compares pixels of the each channel of the source image ROI *pSrc* to a given *value* specified for each channel using the *ippCmpOp* compare operation, and writes the results to a 1-channel Ipp8u image *pDst*. If the result of the compare is true, that is all pixels of all channels satisfy to the specified condition, then the corresponding output pixel is set to an IPP\_MAX\_8U value; otherwise, it is set to 0.

The following code example shows how to use comparison function and create a mask image:

### Example 7-2 Comparing Image to a Constant Value

---

```

IppStatus compare ( void ) {
    Ipp8u x[5*4], y[5*4];
    IppiSize roi = {5,4};
    int i;
    for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;
    return ippiCompareC_8u_C1R ( x, 5, 7, y, 5, roi,
    ippCmpGreater );
}

```

The mask image *y* contains:

```

00 00 00 00 00
00 00 00 FF FF
FF FF FF FF FF
FF FF FF FF FF

```

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

## CompareEqualEps

*Compares two images with floating-point data, testing whether pixel values are equal within a certain tolerance eps.*

---

### Syntax

```
IppStatus ippCompareEqualEps_<mod>(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep,
    IppiSize roiSize, Ipp32f eps);
```

Supported values for *mod* :

32f\_C1R

32f\_C3R

32f\_C4R

32f\_AC4R

### Parameters

<i>pSrc1, pSrc2</i>	Pointers to the source image ROIs.
<i>src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>eps</i>	The tolerance value.

### Description

The function `ippCompareEqualEps` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function tests if the corresponding pixels of ROI in two source images *pSrc1*, *pSrc2* are equal within the tolerance *eps*, and writes the results to a 1-channel `Ipp8u` image *pDst*. If the absolute value of difference of the pixel values in *pSrc1* and *pSrc2* is less than or equal to *eps*,

then the corresponding pixel in *pDst* is set to an `IPP_MAX_8U` value; otherwise the pixel in *pDst* is set to 0. For multi-channel images, the differences for all color channel values of a pixel must be within the tolerance *eps* for the compare condition to be true.

This function processes images with floating-point data only.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>src1Step</i> , <i>src2Step</i> , or <i>dstStep</i> has a zero or negative value.
<code>ippStsEpsValErr</code>	Indicates an error condition if <i>eps</i> has a negative value.

---

## CompareEqualEpsC

*Tests whether floating-point pixel values of an image are equal to a given value within a certain tolerance eps.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
ippStatus ippCompareEqualEpsC_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f value, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

#### Case 2: Operation on multi-channel data

```
ippStatus ippCompareEqualEpsC_<mod>(const Ipp32f* pSrc, int srcStep,
    const Ipp32f value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize,
    Ipp32f eps);
```

Supported values for *mod* :

`32f_C3R`

`32f_AC4R`

```
IppStatus ippiCompareEqualEpsC_32f_C4R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize,
    Ipp32f eps);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>value</i>	The value to compare each pixel to. In case of multi-channel data, an array of separate values (one for each channel).
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>eps</i>	The tolerance value.

## Description

The function `ippiCompareEqualEpsC` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function tests if pixel values of the source image ROI *pSrc* are equal to a given constant *value* within the tolerance *eps*, and writes the results to a 1-channel `Ipp8u` image *pDst*. If the absolute value of difference between the pixel value in *pSrc* and *value* is less than or equal to *eps*, then the corresponding pixel in *pDst* is set to an `IPP_MAX_8U` value; otherwise the pixel in *pDst* is set to 0. For multi-channel images, the differences between all color channel values of a pixel and the respective components of *value* must be within the tolerance *eps* for the compare condition to be true.

This function processes images with floating-point data only.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsEpsValErr</code>	Indicates an error condition if <i>eps</i> has a negative value.



# Morphological Operations

## 8

This chapter describes Intel® Integrated Performance Primitives (Intel® IPP) image processing functions that perform morphological operations on images.

[Table 8-1](#) lists functions described in more detail later in this chapter:

**Table 8-1 Morphological Functions**

Function Base Name	Operation
<a href="#">Dilate3x3</a>	Performs dilation of an image using a 3x3 mask.
<a href="#">Erode3x3</a>	Performs erosion of an image using a 3x3 mask.
<a href="#">Dilate</a>	Performs dilation of an image using a general rectangular mask.
<a href="#">Erode</a>	Performs erosion of an image using a general rectangular mask.
<a href="#">MorphologyInitAlloc</a>	Allocates and initializes morphology state structures for the erosion or dilatation operation.
<a href="#">MorphologyFree</a>	Frees memory allocated for the morphology state structure.
<a href="#">MorphologyInit</a>	Initializes morphology state structures for the erosion or dilatation operation.
<a href="#">MorphologyGetSize</a>	Computes the size of the morphology state structures for the erosion or dilatation operation.
<a href="#">DilateBorderReplicate</a>	Performs dilation of an image.
<a href="#">ErodeBorderReplicate</a>	Performs erosion of an image.
<a href="#">MorphAdvInitAlloc</a>	Allocates and initializes morphology state structure for advanced morphology operations.
<a href="#">MorphAdvFree</a>	Frees memory allocated for the advanced morphology state structure.

**Table 8-1 Morphological Functions (continued)**

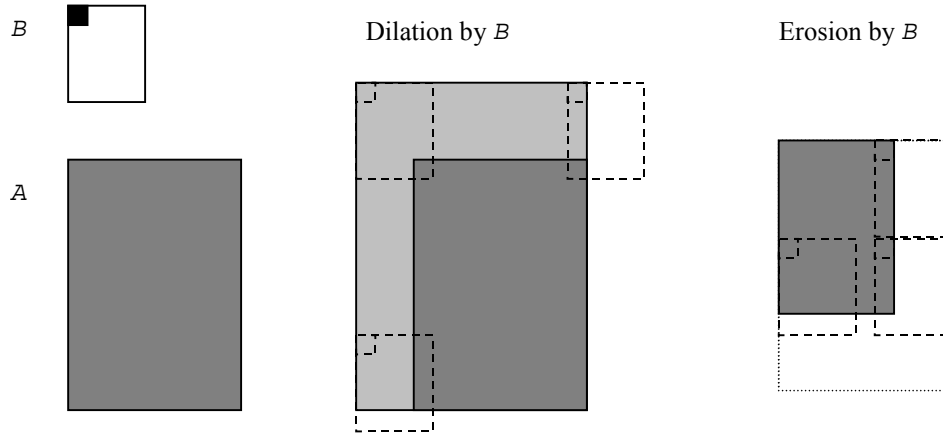
Function Base Name	Operation
<a href="#"><u>MorphAdvInit</u></a>	Initializes morphology state structure for advanced morphology operations.
<a href="#"><u>MorphAdvGetSize</u></a>	Computes the size of the advanced morphology state structure.
<a href="#"><u>MorphOpenBorder</u></a>	Performs opening operation of an image.
<a href="#"><u>MorphCloseBorder</u></a>	Performs closing operation of an image.
<a href="#"><u>MorphTophatBorder</u></a>	Performs top-hat operation of an image.
<a href="#"><u>MorphBlackhatBorder</u></a>	Performs black-hat operation of an image.
<a href="#"><u>MorphGradientBorder</u></a>	Calculates morphological gradient of an image.
<a href="#"><u>MorphGrayInitAlloc</u></a>	Allocates and initializes gray-kernel morphology state structure.
<a href="#"><u>MorphGrayFree</u></a>	Frees memory allocated for the gray-kernel morphology state structure.
<a href="#"><u>MorphGrayInit</u></a>	Initializes gray-kernel morphology state structure.
<a href="#"><u>MorphGrayGetSize</u></a>	Computes the size of the gray-kernel morphology state structure.
<a href="#"><u>GrayDilateBorder</u></a>	Performs gray-kernel dilation of an image.
<a href="#"><u>GrayErodeBorder</u></a>	Performs gray-kernel erosion of an image.
<a href="#"><u>MorphReconstructGetBufferSize</u></a>	Computes the size of the buffer for morphological reconstruction operation
<a href="#"><u>MorphReconstructDilate</u></a>	Reconstructs an image by dilation.
<a href="#"><u>MorphReconstructErode</u></a>	Reconstructs an image by erosion.

Generally, the erosion and dilation smooth the boundaries of objects without significantly changing their area. Opening and closing smooth thin projections or gaps. Morphological operations use a structuring element (SE) that is a user-defined rectangular mask, or for some functions - symmetric 3x3 mask.

In a more general sense, morphological operations involve an image *A* called the *object of interest* and a kernel element *B* called the *structuring element*. The image and structuring element could be in any number of dimensions, but the most common use is with a 2D binary image, or with a 3D gray

scale image. The element  $B$  is most often a square or a circle, but it could be of any shape. Just like in convolution,  $B$  is a kernel or template with an anchor point. [Figure 8-1](#) shows dilation and erosion of object  $A$  by  $B$ . In the figure,  $B$  is rectangular with an anchor point at upper left shown as a dark square.

**Figure 8-1 Dilation and Erosion of  $A$  by  $B$**



Let  $B_t$  is the SE with pixel  $t$  in the anchor position,  $\bar{B}$  is transpose of the SE.

Dilation of binary image  $A$   $\{A(t) = 1, t \in A; 0 - \text{otherwise}\}$  by binary SE  $B$  is

$$A \oplus B = \{t : B_t \cap A \neq \emptyset\}$$

It means that every pixel is in the set, if the intersection is not null. That is, a pixel under the anchor point of  $B$  is marked “on”, if at least one pixel of  $B$  is inside of  $A$ .

Erosion of binary image  $A$  by binary SE  $B$  is

$$A \ominus B = \{t : B_t \subseteq A\}$$

That is, a pixel under the anchor of  $B$  is marked “on”, if  $B$  is entirely within  $A$ .

Generalization of dilation and erosion for the gray-scale image  $A$  and the binary SE  $B$  is

$$A \oplus B = \left\{ \max_{u \in B_t \cap A} A(u) \right\}, \quad A \ominus B = \left\{ \min_{u \in B_t \cap A} A(u) \right\}$$

Generalization of dilation and erosion for the gray-scale image  $A$  and the gray-scale SE  $B$  is

$$A \oplus B = \left\{ \max(A(u) + B(u - t)) \right\}_{u \in B_t \cap A}, \quad A \ominus B = \left\{ \min(A(u) + B(u - t)) \right\}_{u \in B_t \cap A}$$

Opening operation of  $A$  by  $B$  is  $A \circ B = (A \oplus B) \ominus \bar{B}$ .

Closing operation of  $A$  by  $B$  is  $A \bullet B = (A \ominus B) \oplus \bar{B}$ ,

Top-hat operation of  $A$  by  $B$  is  $A - A \circ B$ .

Black-hat operation of  $A$  by  $B$  is  $A \bullet B - A$ .

Morphological gradient of  $A$  by  $B$  is  $A \oplus B - A \ominus B$ .

Morphological reconstruction  $\rho_A(C)$  of an image  $A$  from the image  $C$ ,  $A(t) \geq C(t) \forall t$  by dilation with the mask  $B$  is an image

$$C_k: k = \min_i C_i \equiv C_{i-1}, C_0 = C, C_{i+1}(t) = \min\{(C_i \oplus B)(t), A(t)\}$$

Morphological reconstruction  $\rho_A(C)$  of an image  $A$  from the image  $C$ ,  $A(t) \leq C(t) \forall t$  by erosion with the mask  $B$  is an image

$$C_k: k = \min_i C_i \equiv C_{i-1}, C_0 = C, C_{i+1}(t) = \max\{(C_i \ominus B)(t), A(t)\}$$

[Figure 8-2](#) presents the results of different morphological operations applied to the initial image . In these operations the SE is a matrix of 3x3 size with the following values:

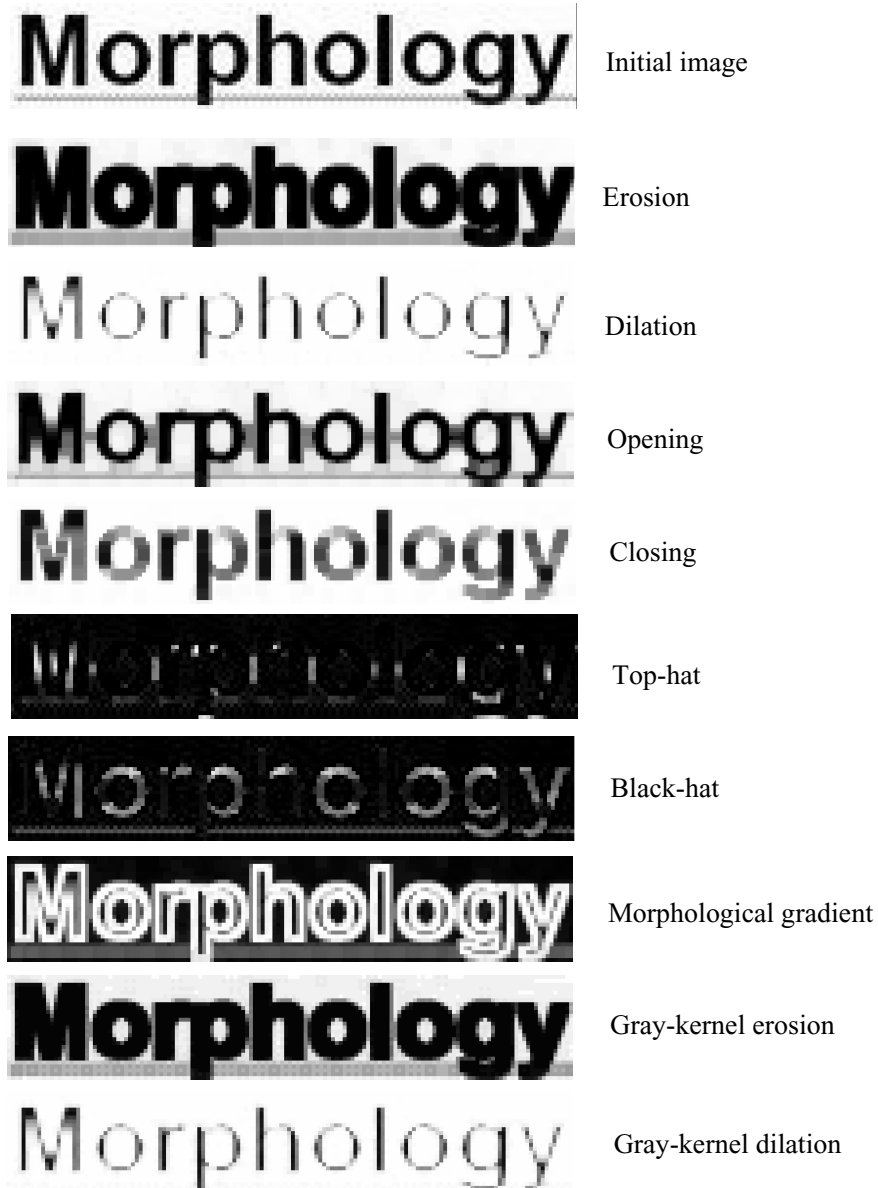
$$\begin{bmatrix} -8 & 0 & -8 \\ 0 & 8 & 0 \\ -8 & 0 & -8 \end{bmatrix} \text{ for common and advanced morphology, and } \begin{bmatrix} -5 & 0 & -5 \\ 0 & 5 & 0 \\ -5 & 0 & -5 \end{bmatrix} \text{ for gray morphology.}$$

The anchor cell in the center cell (1, 1) of the matrix.

---

**Figure 8-2 Morphological Operations Performed by the Intel IPP**

---

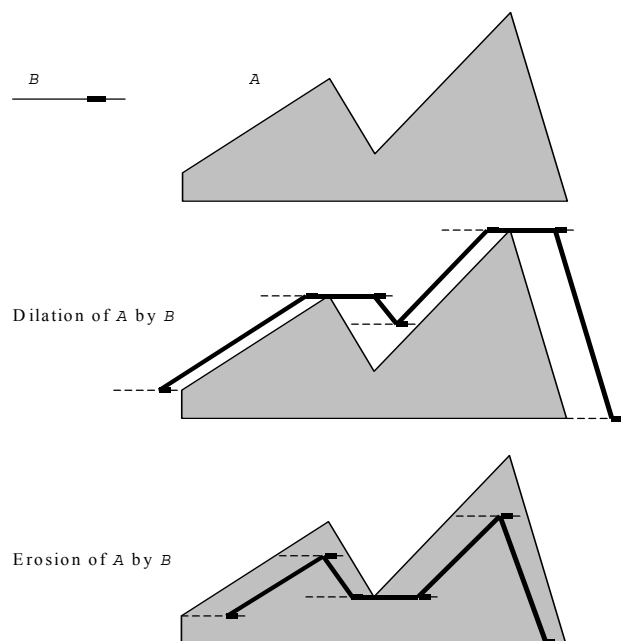


## Flat Structuring Elements for Grayscale Image

Erosion and dilation can be done in 3D space, that is, with gray levels. 3D structuring elements can be used, but the simplest and the best way is to use a flat structuring element  $B$ . [Figure 8-3](#) is a 1D cross section of dilation and erosion of a grayscale image  $A$  by a flat structuring element  $B$ . In the figure,  $B$  has an anchor slightly to the right of the center as shown by the dark mark on  $B$ .

**Figure 8-3** 1D Cross Section of Dilation and Erosion of  $A$  by  $B$

---



In [Figure 8-3](#), dilation is mathematically  $\sup_{y \in B_c} A$  and erosion is  $\inf_{y \in B_c} A$ .

---

## Dilate3x3

*Performs dilation of an image  
using a 3x3 mask.*

---

### Syntax

#### Case 1: Not-in-place operation .

```
IppStatus ippiDilate3x3_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

#### Case 2: In-place operation .

```
IppStatus ippiDilate3x3_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,  
    IppiSize roiSize);
```

Supported values for *mod* :

8u_C1IR	32f_C1IR
8u_C3IR	32f_C3IR
8u_C4IR	32f_C4IR
8u_AC4IR	32f_AC4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROIs for the in-place operation.

<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiDilate3x3` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs dilation of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a symmetric 3x3 mask.

Source and destination images can have different size, but the ROI size is the same for both images. The output pixel is set to the maximum of the corresponding input pixel and its 8 neighboring pixels.

The code example [Example 8-1](#) that shows how to the dilation function is given below:

### Example 8-1 Dilation of a Dot

---

```
IppStatus dilate( void ) {
    Ipp8u x[7*5];
    IppiSize roi = {7,5};
    ippiSet_8u_C1R( 0, x, 7, roi );
    x[2*7+3] = 1;
    roi.width = roi.width - 2;
    roi.height = roi.height - 2;
    return ippiDilate3x3_8u_C1IR( x+7+1, 7, roi );
}
```

The destination image `x` contains:

```
00 00 00 00 00 00 00
00 00 01 01 01 00 00
00 00 01 01 01 00 00
00 00 01 01 01 00 00
00 00 00 00 00 00 00
```

---



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> or <i>srcDstStep</i> has a zero or negative value.

## Erode3x3

*Performs erosion of an image  
using a 3x3 mask.*

### Syntax

#### Case 1: Not-in-place operation.

```
IppStatus ippErode3x3_<mod>(const Ipp<datatype>* pSrc, int srcStep,
                             Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>32f_AC4R</code>

#### Case 2: In-place operation.

```
IppStatus ippErode3x3_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
                             IppiSize roiSize);
```

Supported values for *mod* :

<code>8u_C1IR</code>	<code>32f_C1IR</code>
<code>8u_C3IR</code>	<code>32f_C3IR</code>
<code>8u_C4IR</code>	<code>32f_C4IR</code>
<code>8u_AC4IR</code>	<code>32f_AC4IR</code>

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROIs for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiErode3x3` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs erosion of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a symmetric 3x3 mask.

Source and destination images can have different size, but the ROI size is the same for both images. The output pixel is set to the minimum of the corresponding input pixel and its 8 neighboring pixels.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> or <i>srcDstStep</i> has a zero or negative value.

## Dilate

*Performs dilation of an image  
using a specified mask.*

### Syntax

#### Case 1: Not-in-place operation .

```
IppStatus ippiDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const char* pMask, IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

#### Case 2: In-place operation .

```
IppStatus ippiDilate_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const char* pMask, IppiSize maskSize,
    IppiPoint anchor);
```

Supported values for *mod* :

8u_C1IR	32f_C1IR
8u_C3IR	32f_C3IR
8u_AC4IR	32f_AC4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROIs for the in-place operation.

<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>pMask</i>	Pointer to mask values. Only pixels that correspond to nonzero mask values are taken into account during operation.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

## Description

The function `ippiDilate` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs dilation of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified mask *pMask* of size *maskSize* and alignment *anchor*. Source and destination images can have different size, but the ROI size is the same for both images. The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values. In the four-channel image the alpha channel is not processed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>maskSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> or <i>srcDstStep</i> has a zero or negative value.
<code>ippStsZeroMaskValuesErr</code>	Indicates an error condition if all mask values are equal to zero.

## Erode

*Performs erosion of an image  
using a specified mask.*

### Syntax

#### Case 1: Not-in-place operation .

```
IppStatus ippiErode_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    const char* pMask, IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

#### Case 2: In-place operation .

```
IppStatus ippiErode_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, const char* pMask, IppiSize maskSize,
    IppiPoint anchor);
```

Supported values for *mod* :

8u_C1IR	32f_C1IR
8u_C3IR	32f_C3IR
8u_AC4IR	32f_AC4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROIs for the in-place operation.

<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>pMask</i>	Pointer to mask values. Only pixels that correspond to nonzero mask values are taken into account during operation.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

## Description

The function `ippiErode` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs erosion of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified mask *pMask* of size *maskSize* and alignment *anchor*.

Source and destination images can have different size, but the ROI size is the same for both images (*roiSize*). The output pixel is set to the minimum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values. In the four-channel image the alpha channel is not processed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>maskSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> or <i>srcDstStep</i> has a zero or negative value.
<code>ippStsZeroMaskValuesErr</code>	Indicates an error condition if all mask values are equal to zero.

---

## MorphologyInitAlloc

*Allocates and initializes morphology state structure for erosion or dilation operation.*

---

### Syntax

```
IppStatus ippiMorphologyInitAlloc_<mod>(int roiWidth, const Ipp8u*
    pMask, IppiSize maskSize, IppiPoint anchor, IppiMorphState**
    ppState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<i>roiWidth</i>	Maximal width of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Coordinates of the anchor cell.
<i>ppState</i>	Pointer to the pointer to the morphology state structure.

### Description

The function `ippiMorphologyInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory, initializes the morphology state structure and returns the pointer *ppState* to it. It is used by the functions [ippiDilateBorderReplicate](#) and [ippiErodeBorderReplicate](#) that perform morphological operations on the source image pixels corresponding to non-zero values of the structuring element *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.




---

**WARNING.** The structure can be used to processed image with ROI that does not exceed the specified maximum width *roiWidth*

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with zero or negative value, or if <i>roiWidth</i> is less than 1.
<code>ippStsAnchorErr</code>	Indicates an error if <i>anchor</i> is outside the mask.

---

## MorphologyFree

*Frees memory allocated for the morphology state structure.*

---

### Syntax

```
IppStatus ippiMorphologyFree(IppiMorphState* pState);
```

### Parameters

*pState*                      Pointer to the morphology state structure.

### Description

The function `ippiMorphologyFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function [ippiMorphologyInitAlloc](#) for the morphology state structure *pState*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> is <code>NULL</code> .



---

## MorphologyInit

*Initializes morphology state structure for erosion or dilation operation.*

---

### Syntax

```
IppStatus ippiMorphologyInit_<mod>(int roiWidth, const Ipp8u* pMask,  
    IppiSize maskSize, IppiPoint anchor, IppiMorphState* pState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<i>roiWidth</i>	Maximal width of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Coordinates of the anchor cell.
<i>pState</i>	Pointer to the morphology state structure.

### Description

The function `ippiMorphologyInit` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function initializes the morphology state structure *pState* in the external buffer. Its size should be computed by the function [ippiMorphologyGetSize](#). This structure is used by the functions [ippiDilateBorderReplicate](#) and [ippiErodeBorderReplicate](#) that perform

morphological operations on the source image pixels corresponding to non-zero values of the structuring element (mask) *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.




---

**WARNING.** The structure can be used to processed image with ROI that does not exceed the specified maximum width *roiWidth*

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with zero or negative value, or if <i>roiWidth</i> is less than 1.
<code>ippStsAnchorErr</code>	Indicates an error if <i>anchor</i> is outside the mask.

---

## MorphologyGetSize

*Computes the size of the morphology state structure.*

---

### Syntax

```
IppStatus ippiMorphologyGetSize_<mod>(int roiWidth, const Ipp8u* pMask,
    IppiSize maskSize, int* pSize);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

### Parameters

<i>roiWidth</i>	Maximal width of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.

*pSize* Pointer to the size of the morphology state structure.

## Description

The function `ippiMorphologyGetSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the morphology state structure *pState*. This function should be run prior to the function [ippiMorphologyInit](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with zero or negative value, or if <i>roiWidth</i> is less than 1.

---

# DilateBorderReplicate

*Performs dilation of an image.*

---

## Syntax

```
IppStatus ippiDilateBorderReplicate_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphState* pState);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; following values are possible: <code>ippBorderRep1</code> Replicated border is used.
<i>pState</i>	Pointer to the morphology state structure.

## Description

The function `ippiDilateBorderReplicate` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs dilation of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions [ippiMorphologyInitAlloc](#) or [ippiMorphologyInit](#) beforehand.

The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values.




---

**WARNING.** The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width *roiWidth*.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or if <i>roiSize.width</i> is greater than maximum ROI <i>roiWidth</i> passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

## ErodeBorderReplicate

*Performs erosion of an image.*

### Syntax

```
IppStatus ippiErodeBorderReplicate_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphState* pState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; following values are possible: ippiBorderRepl    Replicated border is used.
<i>pState</i>	Pointer to the morphology state structure.

### Description

The function `ippiErodeBorderReplicate` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs erosion of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions [ippiMorphologyInitAlloc](#) or [ippiMorphologyInit](#) beforehand.

The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values.




---

**WARNING.** The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width *roiWidth*

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or if <i>roiSize.width</i> is greater than maximum ROI <i>roiWidth</i> passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

---

## MorphAdvInitAlloc

*Allocates and initializes morphology state structure for advanced morphology operations.*

---

## Syntax

```
IppStatus ippIMorphAdvInitAlloc_<mod>( IppiMorphAdvState** ppState,
    IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint
    anchor);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

## Parameters

<i>ppState</i>	Pointer to the pointer to the advanced morphology state structure.
<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Coordinates of the anchor cell.

## Description

The function `ippiMorphAdvInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory, initializes the advanced morphology state structure and returns the pointer *ppState* to it. It is used by the functions [ippiMorphOpenBorder](#), [ippiMorphCloseBorder](#), [ippiMorphTophatBorder](#), [ippiMorphBlackhatBorder](#), and [ippiMorphGradientBorder](#) that perform advanced morphological operations (opening, closing, top-hat, black-hat, and gradient) on the source image pixels corresponding to non-zero values of the structuring element *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.




---

**WARNING.** The structure can be used to processed image with ROI that does not exceed the specified maximum width and height *roiSize*

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> or if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error if <i>anchor</i> is outside the mask.

## MorphAdvFree

*Frees memory allocated for the advanced morphology state structure.*

## Syntax

```
IppStatus ippiMorphAdvFree(IppiMorphAdvState* pState);
```

## Parameters

*pState* Pointer to the advanced morphology state structure.

## Description

The function `ippiMorphAdvFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function [`ippiMorphAdvInitAlloc`](#) for the advanced morphology state structure `pState`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

ippStsNullPtrErr	Indicates an error condition if <i>pState</i> is NULL.
------------------	--

## MorphAdvInit

*Initializes morphology state structure for advanced morphology operations.*

## Syntax

```

IppStatus ippiMorphologyInit_<mod>( IppiMorphAdvState* pState, IppiSize
    roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiPoint anchor);

```

Supported values for *mod* :

8u\_C1R      32f\_C1R

8u\_C3R      32f\_C3R

8u\_C4R      32f\_C4R



## Parameters

<i>pState</i>	Pointer to the advanced morphology state structure.
<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Coordinates of the anchor cell.

## Description

The function `ippiMorphAdvInit` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function initializes the advanced morphology state structure *pState* in the external buffer. Its size should be computed by the function [ippiMorphAdvGetSize](#). It is used by the functions [ippiMorphOpenBorder](#), [ippiMorphCloseBorder](#), [ippiMorphTophatBorder](#), [ippiMorphBlackhatBorder](#), and [ippiMorphGradientBorder](#) that perform advanced morphological operations (opening, closing, top-hat, black-hat, and gradient) on the source image pixels corresponding to non-zero values of the structuring element (mask) *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.




---

**WARNING.** The structure can be used to processed image with ROI that does not exceed the specified maximum width and height *roiSize*

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> or if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error if <i>anchor</i> is outside the mask.

## MorphAdvGetSize

*Computes the size of the advanced morphology state structure.*

---

### Syntax

```
IppStatus ippMorphAdvGetSize_<mod>(IppiSize roiSize, const Ipp8u*
    pMask, IppiSize maskSize, int* pSize);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>pSize</i>	Pointer to the size of the advanced morphology state structure.

### Description

The function `ippMorphAdvGetSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the advanced morphology state structure *pState*. This function should be run prior to the function [ippiMorphAdvInit](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> or if <i>roiSize</i> has a field with zero or negative value.

---

## MorphOpenBorder

*Performs opening of an image.*

---

```
IppStatus ippiMorphOpenBorder_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphAdvState* pState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; following values are possible: ippiBorderRepl    Replicated border is used.
<i>pState</i>	Pointer to the advanced morphology state structure.

### Description

The function `ippiMorphOpenBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs opening of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions [ippiMorphAdvInitAlloc](#) or [ippiMorphAdvInit](#) beforehand.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.




---

**WARNING.** The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

---

## MorphCloseBorder

*Performs closing of an image.*

---

```
IppStatus ippIMorphCloseBorder_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphAdvState* pState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; following values are possible: <code>ippBorderRepl</code> Replicated border is used.
<i>pState</i>	Pointer to the advanced morphology state structure.

## Description

The function `ippiMorphCloseBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs closing of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions [ippiMorphAdvInitAlloc](#) or [ippiMorphAdvInit](#) beforehand.

The result is equivalent to successive erosion of the source image by the structured element (mask) and dilation by the reverted structured element.




---

**WARNING.** The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.

<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

---

## MorphTophatBorder

*Performs top-hat operation on an image.*

---

```
IppStatus ippIMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphAdvState* pState);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; following values are possible: <code>ippBorderRepl</code> Replicated border is used.
<i>pState</i>	Pointer to the advanced morphology state structure.

### Description

The function `ippIMorphTophatBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a top-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions [ippiMorphAdvInitAlloc](#) or [ippiMorphAdvInit](#) beforehand.

The result is equivalent to the opening the source image and following subtraction from the initial source image.




---

**WARNING.** The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

---

## MorphBlackhatBorder

*Performs black-hat operation on an image.*

---

```
IppStatus ippiMorphBlackhatBorder_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphAdvState* pState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; following values are possible: ippBorderRep1    Replicated border is used.
<i>pState</i>	Pointer to the advanced morphology state structure.

## Description

The function `ippiMorphBlackhatBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs black-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions [ippiMorphAdvInitAlloc](#) or [ippiMorphAdvInit](#) beforehand.

The result is equivalent to the subtraction of the initial source image from the closed source image.




---

**WARNING.** The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--



<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
<code>ippStsBorderErr</code>	Indicates an error condition if <code>borderType</code> has an illegal value.

---

## MorphGradientBorder

*Calculates morphological gradient of an image.*

---

```
IppStatus ippMorphGradientBorder_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiBorderType borderType, IppiMorphAdvState* pState);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination image ROI.
<code>borderType</code>	Type of border; following values are possible: <code>ippBorderRep1</code> Replicated border is used.

*pState* Pointer to the advanced morphology state structure.

## Description

The function `ippiMorphGradientBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates a morphological gradient of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions [ippiMorphAdvInitAlloc](#) or [ippiMorphAdvInit](#) beforehand.

The result is equivalent to the subtraction of the opened source image from the closed source image.




---

**WARNING.** The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

## MorphGrayInitAlloc

*Allocates and initializes morphology state structure for gray-kernel morphology operations.*

### Syntax

```
IppStatus ippiMorphGrayInitAlloc_8u_C1R(IppiMorphGrayState_8u** ppState,
    IppiSize roiSize, const Ipp32s* pMask, IppiSize maskSize, IppiPoint
    anchor);

IppStatus ippiMorphGrayInitAlloc_32f_C1R(IppiMorphGrayState_32f**
    ppState, IppiSize roiSize, const Ipp32f* pMask, IppiSize maskSize,
    IppiPoint anchor);
```

### Parameters

<i>ppState</i>	Pointer to the pointer to the advanced morphology state structure.
<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Coordinates of the anchor cell.

### Description

The function `ippiMorphGrayInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory, initializes the gray-kernel morphology state structure and returns the pointer *ppState* to it. It is used by the functions [ippiGrayDilateBorder](#) and [ippiGrayErodeBorder](#) that perform gray-kernel dilation and erosion of the source image pixels corresponding to non-zero values of the structuring element *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.



**WARNING.** The structure can be used to processed image with ROI that does not exceed the specified maximum width and height *roiSize*

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> or if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error if <i>anchor</i> is outside the mask.

---

## MorphGrayFree

*Frees memory allocated for the gray-kernel morphology state structure.*

---

### Syntax

```
IppStatus ippMorphGrayFree_8u_C1R(IppiMorphAdvState_8u* pState);
IppStatus ippMorphGrayFree_32f_C1R(IppiMorphAdvState_32f* pState);
```

### Parameters

*pState*                      Pointer to the gray-kernel morphology state structure.

### Description

The function `ippMorphGrayFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function [ippiMorphGrayInitAlloc](#) for the gray-kernel morphology state structure *pState*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> is NULL.

## MorphGrayInit

*Initializes morphology state structure for gray-kernel morphology operations.*

### Syntax

```
IppStatus ippiMorphGrayInit_8u_C1R(IppiMorphGrayState_8u* pState,
    IppiSize roiSize, const Ipp32s* pMask, IppiSize maskSize, IppiPoint
    anchor);

IppStatus ippiMorphGrayInit_32f_C1R(IppiMorphGrayState_32f* pState,
    IppiSize roiSize, const Ipp32f* pMask, IppiSize maskSize, IppiPoint
    anchor);
```

### Parameters

<i>pState</i>	Pointer to the pointer to the gray-kernel morphology state structure.
<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Coordinates of the anchor cell.

### Description

The function `ippiMorphGrayInit` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function initializes the gray-kernel morphology state structure *pState* in the external buffer. Its size should be computed by the function [ippiMorphGrayGetSize](#). It is used by the functions [ippiGrayDilateBorder](#) and [ippiGrayErodeBorder](#) that perform gray-kernel dilation and erosion of the source image pixels corresponding to non-zero values of the structuring element *pMask*. The anchor cell *anchor* is positioned in the arbitrary point in the structuring element and is used for positioning the structuring element.



**WARNING.** The structure can be used to processed image with ROI that does not exceed the specified maximum width and height *roiSize*

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> or if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error if <i>anchor</i> is outside the mask.

---

## MorphGrayGetSize

*Computes the size of the gray-kernel morphology state structure.*

---

### Syntax

```
IppStatus ippMorphGrayGetSize_8u_C1R(IppiSize roiSize, const Ipp32s*
    pMask, IppiSize maskSize, int* pSize);
IppStatus ippMorphAdvGetSize_32f_C1R(IppiSize roiSize, const Ipp32f*
    pMask, IppiSize maskSize, int* pSize);
```

### Parameters

<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
<i>pMask</i>	Pointer to the mask.
<i>maskSize</i>	Size of the mask in pixels.
<i>pSize</i>	Pointer to the size of the advanced morphology state structure.

### Description

The function `ippMorphGrayGetSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the morphology state structure *pState* for gray-kernel dilation and erosion. This function should be run prior to the function [ippMorphGrayInit](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>maskSize</i> or if <i>roiSize</i> has a field with zero or negative value.

---

## GrayDilateBorder

*Performs gray-kernel dilation of an image.*

---

### Syntax

```
IppStatus ippGrayDilateBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphGrayState_8u* pState);

IppStatus ippGrayDilateBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphGrayState_32f* pState);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; following values are possible: <code>ippBorderRep1</code> Replicated border is used.
<i>pState</i>	Pointer to the gray-kernel morphology state structure.

### Description

The function `ippGrayDilateBorder` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs gray-kernel dilation of a rectangular ROI area inside a one-channel 2D image using a specified in the gray-kernel morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions [ippiMorphGrayInitAlloc](#) or [ippiMorphGrayInit](#) beforehand.




---

**WARNING.** The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

---

## GrayErodeBorder

*Performs gray-kernel erosion of an image.*

---

### Syntax

```

IppStatus ippiGrayErodeBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphGrayState_8u* pState);

IppStatus ippiGrayErodeBorder_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, IppiMorphGrayState_32f* pState);

```



## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>borderType</i>	Type of border; following values are possible: <code>ippBorderRepl</code> Replicated border is used.
<i>pState</i>	Pointer to the gray-kernel morphology state structure.

## Description

The function `ippiGrayErodeBorder` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs gray-kernel erosion of a rectangular ROI area inside a one-channel 2D image using a specified in the gray-kernel morphology state structure *pState* mask and anchor cell. This structure must be initialized by the functions [ippiMorphGrayInitAlloc](#) or [ippiMorphGrayInit](#) beforehand.




---

**WARNING.** The function can process only images with ROI that does not exceed the specified by the initialization functions maximum width and height *roiSize*.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .

<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has an illegal value.

---

## MorphReconstructGetBufferSize

*Computes the size of the buffer for morphological reconstruction operation.*

---

```

IppStatus ippMorphReconstructGetBufferSize_8u_C1(IppiSize roiSize, int*
    pSize);
IppStatus ippMorphReconstructGetBufferSize_32f_C1(IppiSize roiSize, int*
    pSize);

```

### Parameters

<i>roiSize</i>	Maximal size of the image ROI in pixels, that can be processed using the buffer.
<i>pSize</i>	Pointer to the size of the buffer.

### Description

The function `ippMorphReconstructBufferGetSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size *pSize* of the buffer for the morphological reconstruction of the source image. This buffer can be used by the functions `ippMorphReconstruct` and `ippMorphReconstruct_32f`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSize</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## MorphReconstructDilate

*Reconstructs an image by dilation.*

---

### Syntax

```
IppStatus ippMorphReconstructDilate_8u_C1IR(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u*
    pBuffer, IppiNorm norm);

IppStatus ippMorphReconstructDilate_32f_C1IR(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f*
    pBuffer, IppiNorm norm);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.				
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.				
<i>pSrcDst</i>	Pointer to the decreased and reconstructed image ROI.				
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the decreased and reconstructed image.				
<i>roiSize</i>	Size of the source and destination image ROI.				
<i>pBuffer</i>	Pointer to the buffer.				
<i>norm</i>	Type of norm to form the mask for dilation; following values are possible: <table data-bbox="565 1084 1380 1146"> <tr> <td><code>ippiNormInf</code></td><td>Infinity norm (8-connectivity, 3x3 rectangular mask).</td></tr> <tr> <td><code>ippiNormL1</code></td><td>L1 norm (4-connectivity, 3x3 cross mask).</td></tr> </table>	<code>ippiNormInf</code>	Infinity norm (8-connectivity, 3x3 rectangular mask).	<code>ippiNormL1</code>	L1 norm (4-connectivity, 3x3 cross mask).
<code>ippiNormInf</code>	Infinity norm (8-connectivity, 3x3 rectangular mask).				
<code>ippiNormL1</code>	L1 norm (4-connectivity, 3x3 cross mask).				

### Description

The function `ippMorphReconstructDilate` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs morphological reconstruction of the decreased source image by dilation [[Vincent93](#)]. The operation is performed in the working buffer whose size should be computed using the function [ippMorphReconstructGetBufferSize](#) beforehand.

This operation allows to detect the regional maximums that can be used as markers for successive watershed segmentation.

The example [Example 8-2](#) shows how the morphological reconstruction can be used to build markers of objects with different brightness. Some value (cap size) is subtracted from the initial image and then the subtracted image is reconstructed to the initial one. Thresholding and opening complete the building of markers. [Figure 8-5](#) shows the results of these operations.

### Example 8-2 Morphological Reconstruction

---

```
IppiMorphAdvState *state;
IppiSize roi, msize={3,3};
IppiPoint anchor={msize.width/2, msize.height/2};
Ipp8u *src, *dst, *img;
Ipp8u *buf, *mask={1,1,1,1,1,1,1,1,1};
Int step, size;

...

ippiMorphReconstructGetBufferSize_8u_C1(roi, &size);
buf = ippsMalloc(size);
ippiMorphAdvInitAlloc_8u_C1R(roi, mask, msize, anchor, &state);

ippiCopy_8u_C1R(src, step, dst, step, roi);
ippiCopy_8u_C1R(src, step, img, step, roi);
// subtract cap size
ippiSubC_8u_C1RSfs(cap, dst, step, roi, 0);
// reconstruct image
ippiMorphReconstructDilate_8u_C1R(src, step, dst, step, roi, buf, norm);
// get caps
ippiSub_8u_C1RSfs(dst, step, img, step, roi, 0);
// caps to white
ippiThreshold_GTVal_8u_C1R(img, step, roi, 0, 255);
// delete noise
ippiMorphOpenBorder_8u_C1R(img, step, dst, step, roi,
                           ipBorderRepl, state);
// revert to get markers
ippiXorC_8u_C1R(0xff, dst, step, roi, 0);

ippiMorphAdvFree(state);
ippsFree(buf);
```

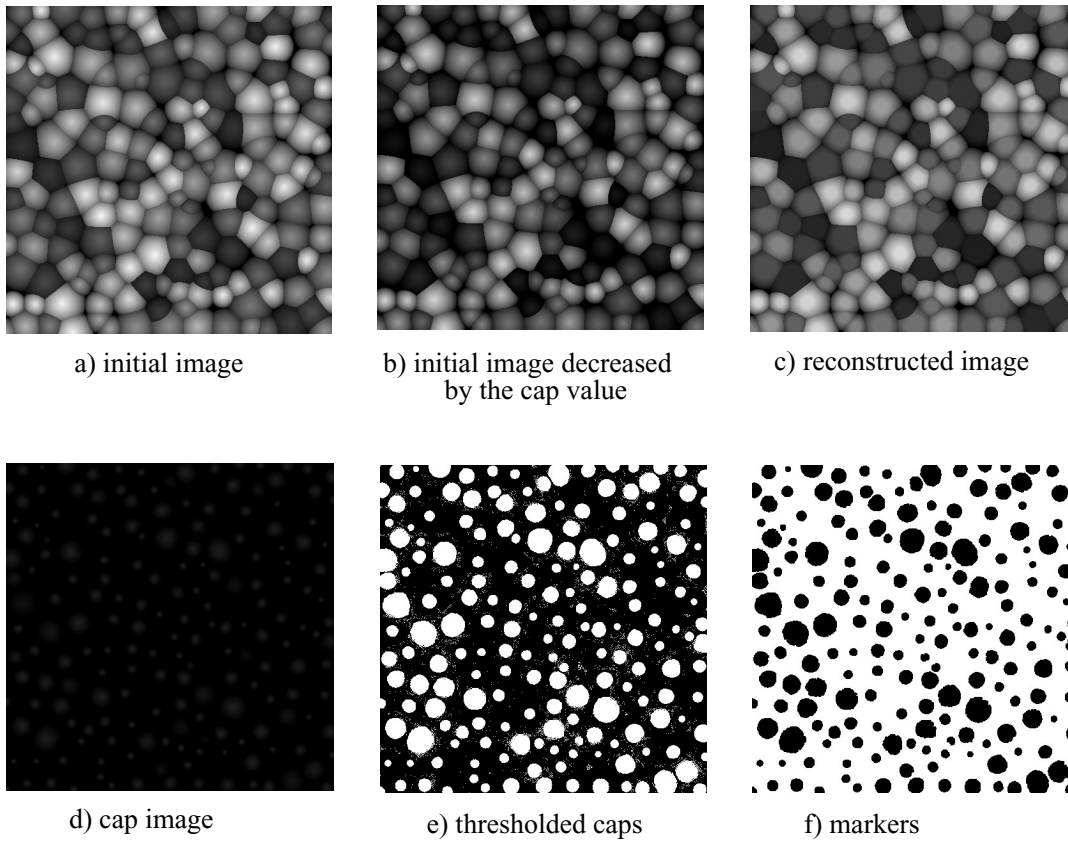
---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>srcDstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .

<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>norm</i> has an illegal value.

**Figure 8-4 Building Markers for Segmentation by the Morphological Reconstruction**



## MorphReconstructErode

*Reconstructs an image by erosion.*

---

### Syntax

```

IppStatus ippiMorphReconstructErode_8u_C1IR(const Ipp8u* pSrc, int
    srcStep, Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u*
    pBuf, IppiNorm norm);

IppStatus ippiMorphReconstructErode_32f_C1IR(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f*
    pBuf, IppiNorm norm);

```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.				
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.				
<i>pSrcDst</i>	Pointer to the increased image ROI.				
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the increased image.				
<i>roiSize</i>	Size of the source and destination image ROI.				
<i>pBuffer</i>	Pointer to the buffer.				
<i>norm</i>	Type of norm to form the mask for erosion; following values are possible: <table data-bbox="633 1076 1443 1146"> <tr> <td><code>ippiNormInf</code></td><td>Infinity norm (8-connectivity, 3x3 rectangular mask).</td></tr> <tr> <td><code>ippiNormL1</code></td><td>L1 norm (4-connectivity, 3x3 cross mask).</td></tr> </table>	<code>ippiNormInf</code>	Infinity norm (8-connectivity, 3x3 rectangular mask).	<code>ippiNormL1</code>	L1 norm (4-connectivity, 3x3 cross mask).
<code>ippiNormInf</code>	Infinity norm (8-connectivity, 3x3 rectangular mask).				
<code>ippiNormL1</code>	L1 norm (4-connectivity, 3x3 cross mask).				

### Description

The function `ippiMorphReconstructErode` is declared in the `ippcv.h`. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs morphological reconstruction of the increased source image by erosion [[Vincent93](#)]. The operation is performed in the working buffer whose size should be computed using the function [ippiMorphReconstructGetBufferSize](#) beforehand.

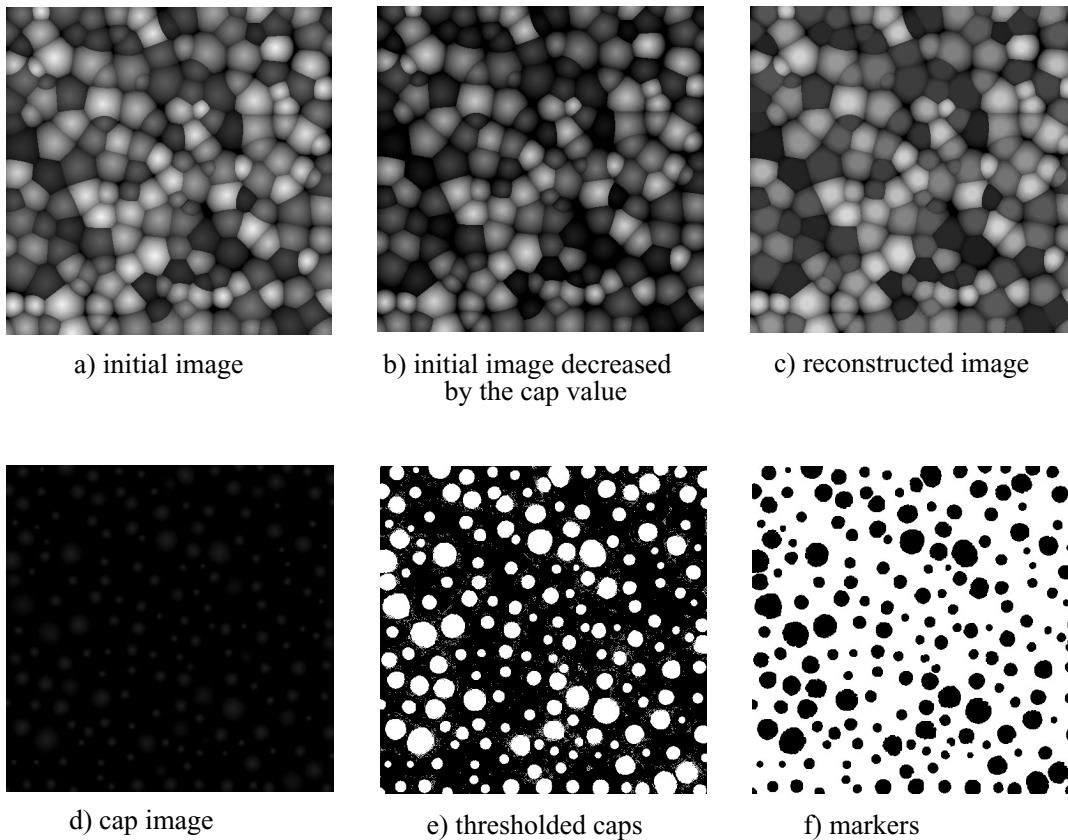
This operation allows to detect the regional minimums that can be used as markers for successive watershed segmentation.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>srcDstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images..
<code>ippStsBadArgErr</code>	Indicates an error condition if <code>norm</code> has an illegal value.

**Figure 8-5 Building Markers for Segmentation by the Morphological Reconstruction**

---





# Filtering Functions

## 9

This chapter describes Intel IPP image processing functions that perform linear and non-linear filtering operations on an image.

Filtering can be used in a variety of image processing operations; for example, edge detection, blurring, noise removal, and feature detection.

[Table 9-1](#) lists functions described in more detail later in this chapter:

**Table 9-1      Filtering functions**

Function Base Name	Operation
<a href="#">FilterBox</a>	Blurs an image using a box filter
<a href="#">SumWindowRow</a>	Sums pixel values in the row mask applied to the image.
<a href="#">SumWindowColumn</a>	Sums pixel values in the column mask applied to the image.
<a href="#">FilterMin</a>	Filters an image using a <i>min</i> filter
<a href="#">FilterMax</a>	Filters an image using a <i>max</i> filter
<a href="#">FilterMinGetBufferSize</a>	Computes the size of the working buffer for the minimum filter
<a href="#">FilterMaxGetBufferSize</a>	Computes the size of the working buffer for the maximum filter
<a href="#">FilterMinBorderReplicate</a>	Filters an image using a <i>min</i> filter with border replication
<a href="#">FilterMaxBorderReplicate</a>	Filters an image using a <i>max</i> filter with border replication
<b>Median Filters</b>	
<a href="#">FilterMedian</a>	Applies a median filter to an image

**Table 9-1 Filtering functions (continued)**

Function Base Name	Operation
<a href="#">FilterMedianHoriz</a>	Filters an image using a median filter with a horizontal mask
<a href="#">FilterMedianVert</a>	Filters an image using a median filter with a vertical mask
<a href="#">FilterMedianCross</a>	Filters an image using a cross median filter
<a href="#">FilterMedianColor</a>	Applies a color median filter to an image
<b>General Linear Filters</b>	
<a href="#">Filter</a>	Filters an image using a general rectangular convolution kernel
<a href="#">Filter32f</a>	Filters an image with integer data using a floating-point rectangular convolution kernel
<b>Separable Filters</b>	
<a href="#">FilterColumn</a>	Filters an image using an integer column convolution kernel
<a href="#">FilterColumn32f</a>	Filters an image using a floating-point column convolution kernel
<a href="#">FilterRow</a>	Filters an image using an integer row convolution kernel
<a href="#">FilterRow32f</a>	Filters an image using a floating-point row convolution kernel
<a href="#">FilterRowBorderPipelineGetBufferSize</a>	Compute the size of working buffer for rows filter with border.
<a href="#">FilterRowBorderPipelineGetBufferSize_Low</a>	Compute the size of working buffer for the Low-flavor of row filters.
<a href="#">FilterColumnPipelineGetBufferSize</a>	Compute the size of working buffer for columns filter with border
<a href="#">FilterColumnPipelineGetBufferSize_Low</a>	Compute the size of working buffer for the Low-flavor of column filters
<a href="#">FilterRowBorderPipeline</a>	Applies the filter with border to image rows
<a href="#">FilterRowBorderPipeline_Low</a>	Applies the Low-flavor filter with border to image rows
<a href="#">FilterColumnPipeline</a>	Applies the filter with border to image columns
<a href="#">FilterColumnPipeline_Low</a>	Applies the Low-flavor filter with border to image columns

**Table 9-1**      **Filtering functions (continued)**

Function Base Name	Operation
<b>Wiener Filters</b>	
<a href="#"><u>FilterWienerGetBufferSize</u></a>	Computes the size of the external buffer for ippiFilterWiener function.
<a href="#"><u>FilterWiener</u></a>	Filters an image using the Wiener algorithm
<b>Convolution</b>	
<a href="#"><u>ConvFull</u></a>	Performs full convolution of two images
<a href="#"><u>ConvValid</u></a>	Performs valid convolution of two images
<b>Deconvolution</b>	
<a href="#"><u>DeconvFFTInitAlloc</u></a>	Allocates and initializes state structure for FFT deconvolution
<a href="#"><u>DeconvFFTFree</u></a>	Frees memory allocated for the FFT deconvolution state structure
<a href="#"><u>DeconvFFT</u></a>	Performs FFT deconvolution of an image
<a href="#"><u>DeconvLRInitAlloc</u></a>	Allocates and initializes state structure for LR deconvolution
<a href="#"><u>DeconvLRFree</u></a>	Frees memory allocated for the LR deconvolution state structure
<a href="#"><u>DeconvLR</u></a>	Performs LR deconvolution of an image
<b>Fixed Filters</b>	
<a href="#"><u>FilterPrewittHoriz</u></a>	Filters an image using a horizontal Prewitt operator
<a href="#"><u>FilterPrewittVert</u></a>	Filters an image using a vertical Prewitt operator
<a href="#"><u>FilterScharrHoriz</u></a>	Filters an image using a horizontal Scharr operator
<a href="#"><u>FilterScharrVert</u></a>	Filters an image using a vertical Scharr operator
<a href="#"><u>FilterSobelHoriz</u></a>	Filters an image using a horizontal Sobel operator
<a href="#"><u>FilterSobelHorizMask</u></a>	
<a href="#"><u>FilterSobelVert</u></a>	Filters an image using a vertical Sobel operator
<a href="#"><u>FilterSobelVerMaskt</u></a>	
<a href="#"><u>FilterSobelHorizSecond</u></a>	Filters an image using a second derivative horizontal Sobel operator.
<a href="#"><u>FilterSobelVertSecond</u></a>	Filters an image using a second derivative vertical Sobel operator.
<a href="#"><u>FilterSobelCross</u></a>	Filters an image using a second cross derivative Sobel operator.

**Table 9-1**      **Filtering functions (continued)**

Function Base Name	Operation
<a href="#">FilterRobertsDown</a>	Filters an image using a horizontal Roberts cross-gradient operator
<a href="#">FilterRobertsUp</a>	Filters an image using a vertical Roberts cross-gradient operator
<a href="#">FilterLaplace</a>	Filters an image using a Laplacian kernel
<a href="#">FilterGauss</a>	Filters an image using a Gaussian kernel
<a href="#">FilterHipass</a>	Applies a highpass filter to an image
<a href="#">FilterLowpass</a>	Applies a lowpass filter to an image
<a href="#">FilterSharpen</a>	Applies a sharpening filter to an image
<b>Fixed Filters with Borders</b>	
<a href="#">FilterScharrHorizGetBufferSize</a>	Computes the size of the external buffer for the horizontal Scharr filter with border.
<a href="#">FilterScharrVertGetBufferSize</a>	Computes the size of the external buffer for the vertical Scharr filter with border.
<a href="#">FilterSobelHorizGetBufferSize</a>	Computes the size of the external buffer for the horizontal Sobel filter with border.
<a href="#">FilterSobelVertGetBufferSize,</a> <a href="#">FilterSobelNegVertGetBufferSize</a>	Computes the size of the external buffer for the vertical Sobel filter with border.
<a href="#">FilterSobelHorizSecondGetBufferSize</a>	Computes the size of the external buffer for the second derivative horizontal Sobel filter with border.
<a href="#">FilterSobelVertSecondGetBufferSize</a>	Computes the size of the external buffer for the second derivative vertical Sobel filter with border.
<a href="#">FilterSobelCrossGetBufferSize</a>	Computes the size of the external buffer for the cross Sobel filter with border.
<a href="#">FilterLaplacianGetBufferSize</a>	Computes the size of the external buffer for the Laplace filter with border.
<a href="#">FilterLowpassGetBufferSize</a>	Computes the size of the external buffer for the lowpass filter with border.
<a href="#">GenSobelKernel</a>	Computes kernel for the Sobel filter to an image.
<a href="#">FilterScharrHorizBorder</a>	Applies horizontal Scharr filter with border to an image.
<a href="#">FilterScharrVertBorder</a>	Applies vertical Scharr filter with border to an image.
<a href="#">FilterSobelHorizBorder</a>	Applies horizontal Sobel filter with border to an image.
<a href="#">FilterSobelVertBorder,</a> <a href="#">FilterSobelNegVertBorder,</a>	Applies vertical Sobel filter with border to an image.

**Table 9-1**      **Filtering functions (continued)**

Function Base Name	Operation
<a href="#"><u>FilterSobelHorizSecondBorder</u></a>	Applies horizontal (second derivative) Sobel filter with border to an image.
<a href="#"><u>FilterSobelVertSecondBorder</u></a>	Applies vertical (second derivative) Sobel filter with border to an image.
<a href="#"><u>FilterSobelCrossBorder</u></a>	Applies second derivative cross Sobel filter with border to an image.
<a href="#"><u>FilterLaplacianBorder</u></a>	Applies Laplacian filter with border to an image.
<a href="#"><u>FilterLowpassBorder</u></a>	Applies lowpass filter with border to an image.

Most of filtering function operate with ROI (see [Regions of Interest in Intel IPP](#)). The size of the source image ROI is equal to `dstRoiSize` the destination image ROI size. Most of the functions use different source and destination image buffers, that is, they are not in-place.

## Borders

Filtering functions described later in this chapter perform neighborhood operations (see [Neighborhood operations](#) in chapter 2). They operate on the assumption that for each pixel being processed, all referred neighborhood pixels necessary for the operation are also available.

The neighborhood for each given pixel is defined by the filter kernel (or mask) size and anchor cell position. Anchor cell (see [Figure 9-1 a](#)) is a fixed cell within the kernel, which is used for positioning the kernel with respect to the currently processed pixel of the source image. Specifically, the kernel is placed on the image in such a way that the anchor cell coincides with the input pixel.

As Fig. 9-1b illustrates, if the input pixel is near the horizontal or vertical edge of the image, the overlaid kernel may refer to neighborhood pixels that do not exist within the source image (that is, are located outside the image area).

The set of all boundary source image pixels that require such non-existent pixels to complete the neighborhood operation for the given kernel and anchor is shaded yellow in [Figure 9-1 b](#)), while the collection of all scanned external pixels (called *border* pixels) is shaded gray.

Hence, if you want to apply some filtering function to the whole source image, you need to first figure out what additional border pixels will be required for the operation, and then define in one way or another these non-existent pixels. To do this, you may either use Intel IPP functions

[ippiCopyConstBorder](#), [ippiCopyReplicateBorder](#), or [ippiCopyWrapBorder](#) which fill the border of the extended image with the pixel values that you define, or apply your own extension method.




---

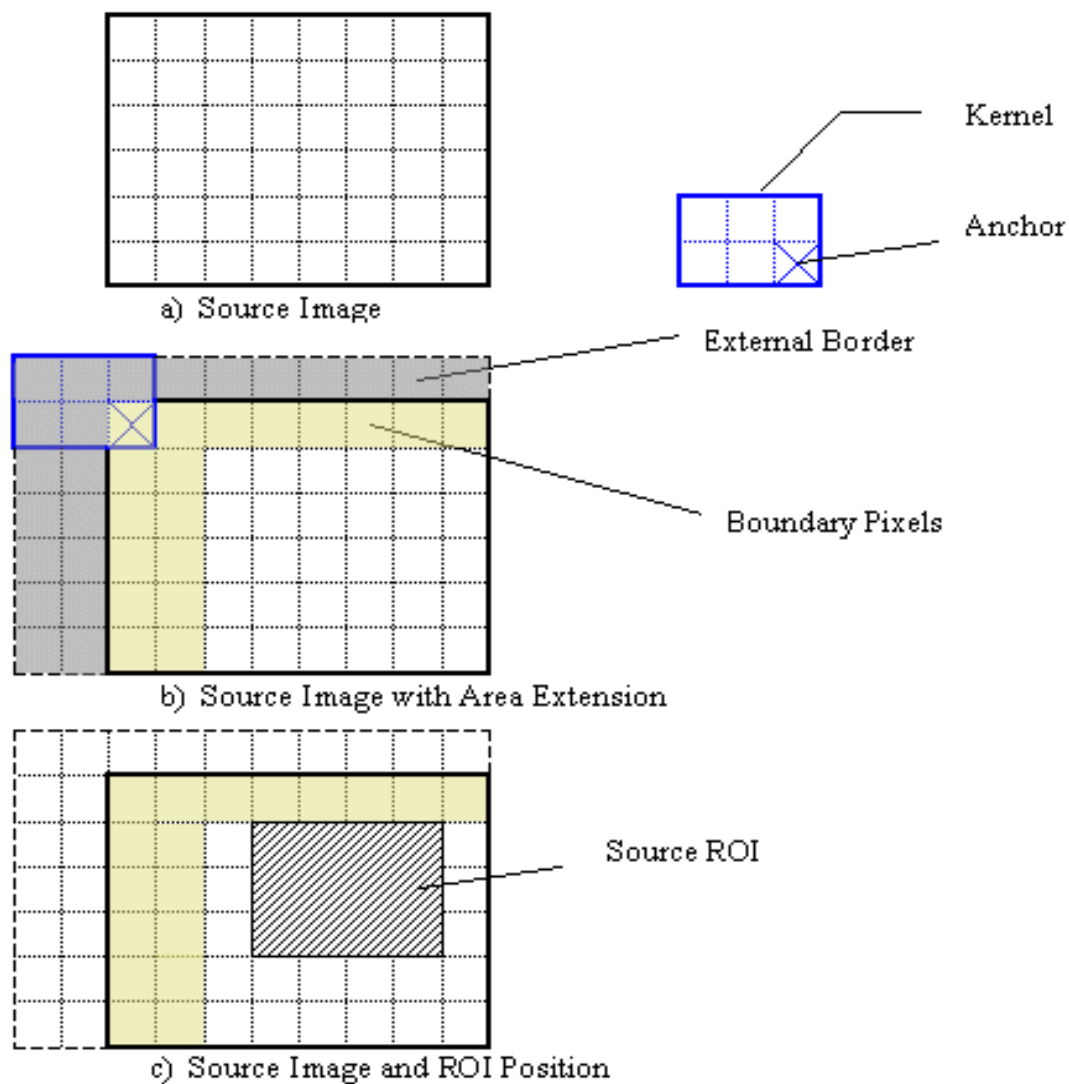
**WARNING.** If the required border pixels are not defined prior to the filtering function call, you may get memory violation errors.

---

On the other hand, if the filtering operation is to be carried out on a part of the source image, or ROI (see [Regions of Interest in Intel IPP](#) in chapter 2), then the necessity of extending the image area with border pixels depends upon the ROI size and position within the image. It can be readily seen ([Figure 9-1 c](#)) that if ROI does not cover yellow (internal boundary) pixels, then no external pixels will be scanned, and border extension is not required.

If boundary pixels are part of ROI, then some area extension of the source image is still necessary.

Figure 9-1 Borders for Neighborhood Operations



The bottom-line is that to provide valid results of filtering operations, the application must check that ROI parameters passed to the filtering function have such values that all required neighborhood pixels actually exist in the image and define the missing pixels when necessary.

---

## FilterBox

*Blurs an image using a simple box filter.*

---

### Syntax

#### Case 1: Not-in-place operation.

```
IppStatus ippiFilterBox_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

#### Case 2: In-place operation.

```
IppStatus ippiFilterBox_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod* :

8u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16s_AC4IR	32f_AC4IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.



<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>roiSize</i>	Size of the source and destination ROI in pixels for the in-place operation.
<i>pSrcDst</i>	Pointer to the source and destination image ROIs for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

## Description

The function `ippiFilterBox` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the output image as the average of all the input image pixels in the rectangular neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of smoothing or blurring the input image. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> or <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> , <i>dstStep</i> , or <i>srcDstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with zero or negative value, or (for in-place flavors) if one of the <i>maskSize</i> fields is greater than or equal to the corresponding field of the <i>roiSize</i> .
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>anchor</i> is outside the mask size.

## SumWindowRow

*Sums pixel values in the row mask applied to the image.*

---

### Syntax

```
IppStatus ippiSumWindowRow_<mod>(const Ipp<srcDatatype>* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int
    maskSize, int xAnchor);
```

Supported values for *mod* :

8u32f_C1R	16s32f_C1R
8u32f_C3R	16s32f_C3R
8u32f_C4R	16s32f_C4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>maskSize</i>	Size of the horizontal row mask in pixels.
<i>xAnchor</i>	Anchor cell specifying the row mask alignment with respect to the position of the input pixel.

### Description

The function `ippiSumWindowRow` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image ROI *pDst* as the sum of all the source image pixels in the horizontal row mask of size *maskSize* with the anchor cell *xAnchor* at the corresponding pixel in the source image ROI *pSrc*. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>xAnchor</i> is outside the mask size.

---

## SumWindowColumn

*Sums pixel values in the column mask applied to the image.*

---

### Syntax

```
IppStatus ippSumWindowColumn_<mod>(const Ipp<srcDatatype>* pSrc, int  
    srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int  
    maskSize, int yAnchor);
```

Supported values for *mod* :

<code>8u32f_C1R</code>	<code>16s32f_C1R</code>
<code>8u32f_C3R</code>	<code>16s32f_C3R</code>
<code>8u32f_C4R</code>	<code>16s32f_C4R</code>

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>maskSize</i>	Size of the vertical column mask in pixels.
<i>anchor</i>	Anchor cell specifying the column mask alignment with respect to the position of the input pixel.

## Description

The function `ippiSumWindowColumn` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image ROI `pDst` as the sum of all the source image pixels in the vertical row mask of size `maskSize` with the anchor cell `yxAnchor` at the corresponding pixel in the source image ROI `pSrc`. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <code>maskSize</code> has a field with zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error condition if <code>yAnchor</code> is outside the mask size.

---

## FilterMin

*Applies the 'min' filter to an image.*

---

## Syntax

```
IppStatus ippiFilterMin_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiSize maskSize, IppiPoint anchor);
```

Supported values for `mod` :

<code>8u_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>16s_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

## Parameters

`pSrc`                      Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

## Description

The function `ippiFilterMin` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image to the minimum value of all the source image pixel values in the neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of decreasing the contrast in the image.

The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

The [Example 9-1](#) illustrates the use of the `ippiFilterMin` function. Note that this function operates on the assumption that all neighborhood pixels needed from outside the ROI are available.

### Example 9-1 Applying the Min Filter

---

```

IppStatus filterMin( void ) {
    Ipp8u x[5*4], y[5*4]={0};
    IppiSize img={5,4}, roi={3,2}, mask={3,3};
    IppiPoint anchor = {1,1};
    ippiSet_8u_C1R( 3, x, 5, img );
    /// set a hole (1) at row 1 and column 1.
    /// The value will be extended on filter
    /// mask area, depending on anchor
    x[5+1] = 1;
    /// ROI is inside the image.
    /// Offset pointers to jump at the ROI start
    return ippiFilterMin_8u_C1R( x+6, 5, y+6, 5, roi, mask,
    anchor );
}

```

The destination image *y* contains:

```

00 00 00 00 00
00 01 01 03 00
00 01 01 03 00
00 00 00 00 00

```

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>anchor</i> is outside the mask size.

## FilterMax

*Applies the 'max' filter to an image.*

### Syntax

```
IppStatus ippiFilterMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

### Description

The function `ippiFilterMax` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image to the maximum value of all the source image pixel values in the neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of increasing the contrast in the image.

The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with zero or negative value.
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>anchor</i> is outside the mask size.

---

## FilterMinGetBufferSize

*Computes the size of the working buffer for the minimum filter.*

---

### Syntax

```
IppStatus ippiFilterMinGetBufferSize_<mod>(int roiWidth, IppiSize
    maskSize, int* pBufferSize);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

### Parameters

<i>roiWidth</i>	Image width (in pixels).
<i>maskSize</i>	Size of the mask in pixels.
<i>pBufferSize</i>	Pointer to the computed size of the buffer.



## Description

The function `ippiFilterMinGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for [ippiFilterMinBorderReplicate](#) function. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than `roiWidth`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <code>pBufferSize</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskSize</code> has a field with zero or negative value, or if <code>roiWidth</code> is less than 1.

---

## FilterMaxGetBufferSize

*Computes the size of the working buffer for the maximum filter.*

---

## Syntax

```
IppStatus ippiFilterMaxGetBufferSize_<mod>(int roiWidth, IppiSize  
    maskSize, int* pBufferSize);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

## Parameters

<code>roiWidth</code>	Image width (in pixels).
<code>maskSize</code>	Size of the mask in pixels.
<code>pBufferSize</code>	Pointer to the computed size of the buffer.

## Description

The function `ippiFilterMaxGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the working buffer required for [ippiFilterMaxBorderReplicate](#) function. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than `roiWidth`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <code>pBufferSize</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskSize</code> has a field with zero or negative value, or if <code>roiWidth</code> is less than 1.

---

## FilterMinBorderReplicate

*Applies the 'min' filter with border replication to an image.*

---

## Syntax

```
IppStatus ippiFilterMinBorderReplicate_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiSize maskSize, IppiPoint anchor, Ipp8u* pBuffer);
```

Supported values for `mod` :

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
<i>pBuffer</i>	Pointer to the working buffer.

## Description

The function `ippiFilterMinBorderReplicate` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image to the minimum value of all the source image pixel values in the neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of decreasing the contrast in the image.

The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. Border pixels are chosen according to the [Figure 4-2](#) (border replication). The function requires the external buffer *pBuffer*, its size should be previously computed by the function [ippiFilterMinGetBufferSize](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>maskSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsAnchorErr</code>	Indicates an error if <i>anchor</i> is outside the mask.

## FilterMaxBorderReplicate

*Applies the 'max' filter with border replication to an image.*

---

### Syntax

```
IppStatus ippiFilterMaxBorderReplicate_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
    IppiSize maskSize, IppiPoint anchor, Ipp8u* pBuffer);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
<i>pBuffer</i>	Pointer to the working buffer.

### Description

The function `ippiFilterMaxBorderReplicate` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the destination image to the maximum value of all the source image pixel values in the neighborhood of size *maskSize* with the anchor cell at that pixel. This has the effect of increasing the contrast in the image.

The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. Border pixels are chosen according to the [Figure 4-2](#) (border replication). The function requires the external buffer *pBuffer*, its size should be previously computed by the function [ippiFilterMaxGetBufferSize](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> or <i>maskSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsAnchorErr</code>	Indicates an error if <i>anchor</i> is outside the mask.

## Median Filters

The median filter functions perform non-linear filtering of a source image data.

These functions use either an arbitrary rectangular mask, or the following predefined masks of the `IppiMaskSize` type to filter an image:

<code>ippMaskSize3x1</code>	Horizontal mask of length 3
<code>ippMaskSize5x1</code>	Horizontal mask of length 5
<code>ippMaskSize1x3</code>	Vertical mask of length 3
<code>ippMaskSize3x3</code>	Square mask of size 3
<code>ippMaskSize1x5</code>	Vertical mask of length 5
<code>ippMaskSize5x5</code>	Square mask of size 5

The size of the neighborhood and coordinates of the anchor cell in the neighborhood depend on the `mask` mean. [Table 9-2](#) lists the mask types with the corresponding neighborhood sizes and anchor cell coordinates. Note that in mask names the mask size is indicated in (*XY*) order. The anchor cell is specified by its coordinates `anchor.x` and `anchor.y` in the coordinate system associated with the top left corner of the mask

**Table 9-2 Median Filter Mask, Neighborhood and Anchor Cell**

Mask	Neighborhood Size		Anchor Cell
	Columns	Rows	
<code>ippMaskSize3x1</code>	3	1	[1, 0]
<code>ippMaskSize5x1</code>	5	1	[2, 0]
<code>ippMaskSize1x3</code>	1	3	[0, 1]
<code>ippMaskSize3x3</code>	3	3	[1, 1]
<code>ippMaskSize1x5</code>	1	5	[0, 2]
<code>ippMaskSize5x5</code>	5	5	[2, 2]

Median filters have the effect of removing the isolated intensity spikes and can be used to achieve noise reduction in an image.

For details on algorithms used in Intel IPP for median filtering, see [\[APMF\]](#).

---

## FilterMedian

*Filters an image using a median filter.*

---

### Syntax

```
IppStatus ippiFilterMedian_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiSize maskSize, IppiPoint anchor);
```

Supported values for *mod* :

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R
8u_AC4R	16s_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

### Description

The function `ippiFilterMedian` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the top left corner of the kernel. The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

The [Example 9-2](#) illustrates median filtering. Note that this filter removes noise and does not cut out signal brightness drops, as an averaging filter does.

---

#### Example 9-2 Applying the Median Filter to an Image

---

```
IppStatus filterMedian( void ) {
    IppiPoint anchor = { 1,1 };
    Ipp8u x[5*4], y[5*4]={0};
    IppiSize img={3,4}, roi={3,2}, mask={3,3};
    ippiSet_8u_C1R( 0x10, x, 5, img );
    /// raise the level of the signal,
    /// The edge will not be destroyed by the filter
    img.width = 5-3;
    ippiSet_8u_C1R( 0x40, x+3, 5, img );
    /// a spike, will be filtered
    x[5+1] = 0;
    /// roi is inside the image.
    /// Offset pointers to jump at the roi's beginning
    return ippiFilterMedian_8u_C1R( x+6, 5, y+6, 5, roi, mask,
    anchor );
}
```

The destination image *y* contains:

```
00 00 00 00 00
00 10 10 40 00
00 10 10 40 00
00 00 00 00 00
```

---

#### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.



---

<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>maskSize</i> has a field with zero, negative, or even value.
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>anchor</i> is outside the mask size.

---

## FilterMedianHoriz

*Filters an image using a horizontal median filter.*

---

### Syntax

```
IppStatusippiFilterMedianHoriz_<mod>(constIpp<datatype>*pSrc, intsrcStep,
    Ipp<datatype>*pDst, intdstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>16s_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>
<code>8u_C4R</code>	<code>16s_C4R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

### Description

The function `ippiFilterMedianHoriz` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The horizontal size of the neighborhood and the anchor cell coordinates depend on the *mask* mean, which can be either `ippMskSize3x1` or `ippMskSize5x1` (see [Table 9-2](#)). The function is used on the assumption that the pixels outside of the source image ROI exist along the distance of half the mask size. It means that the application program should provide appropriate values for the *pSrc* and *dstRoiSize* arguments, or define additional border pixels (see [Borders](#)). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

---

## FilterMedianVert

*Filters an image using a vertical median filter.*

---

### Syntax

```
IppStatus ippiFilterMedianVert_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>16s_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>
<code>8u_C4R</code>	<code>16s_C4R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>

### Parameters

*pSrc*                                      Pointer to the source image ROI.

---

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

## Description

The function `ippiFilterMedianVert` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The vertical size of the neighborhood and the anchor cell coordinates depend on the *mask* mean, which can be either `ippMskSize1x3` or `ippMskSize1x5` (see [Table 9-2](#)). The function is used on the assumption that the pixels outside of the source image ROI exist along the distance of half the mask size. It means that the application program should provide appropriate values for the *pSrc* and *dstRoiSize* arguments, or define additional border pixels (see [Borders](#)). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

## FilterMedianCross

*Filters an image using a cross median filter.*

---

### Syntax

```
IppStatusippiFilterMedianCross_<mod>(constIpp<datatype>*pSrc,intsrcStep,
    Ipp<datatype>*pDst,intdstStep,IppiSize dstRoiSize,IppiMaskSize mask);
```

Supported values for *mod* :

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_AC4R	16s_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

### Description

The function `ippiFilterMedianCross` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The neighborhood is determined by the square mask of the predefined size, which can be either `ippMskSize3x3` or `ippMskSize5x5` (see [Table 9-2](#)). The function is used on the assumption that the pixels outside of the source image ROI exist along the distance of half the mask size. It means that the application program should provide appropriate values for the *pSrc* and *dstRoiSize* arguments, or define additional border pixels (see [Borders](#)). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

---

## FilterMedianColor

*Filters an image using a color median filter.*

---

### Syntax

```
IppStatusippiFilterMedianColor_<mod>(constIpp<datatype>*pSrc,intsrcStep,  
Ipp<datatype>*pDst,intdstStep,IppiSizedstRoiSize,IppiMaskSizemask);
```

Supported values for *mod* :

<code>8u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

## Description

The function `ippiFilterMedianColor` is declared in the `ippi.h` file.

When applied to a color image, the previously described median filtering functions process color planes of an image separately, and as a result any correlation between color components is lost. If you want to preserve this information, use the `ippiFilterMedianColor` function instead. For each input pixel, this function computes differences between red (R), green (G), and blue (B) color components of pixels in the *mask* neighborhood and the input pixel. The distance between the input pixel *i* and the neighborhood pixel *j* is formed as the sum of absolute values

$$\text{abs}(R(i) - R(j)) + \text{abs}(G(i) - G(j)) + \text{abs}(B(i) - B(j))$$

After scanning the entire neighborhood, the function sets the output value for pixel *i* as the value of the neighborhood pixel with the smallest distance to *i*.

The function `ippiFilterMedianColor` supports square masks of size either `ippMskSize3x3` or `ippMskSize5x5` and processes color images only. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

This function operates with ROI (see [Regions of Interest in Intel IPP](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

## General Linear Filters

These functions use a general rectangular kernel to filter an image. The kernel is a matrix of signed integers or single-precision real values. For each input pixel, the kernel is placed on the image in such a way that the fixed anchor cell within the kernel coincides with the input pixel. The anchor cell is usually a geometric center of the kernel, but can be skewed with respect to the geometric center.

A pointer to an array of kernel values is passed to filtering functions. These values are read in row-major order starting from the top left corner. There should be exactly `kernelSize.width * kernelSize.height` entries in this array. The anchor cell is specified by its coordinates `anchor.x` and `anchor.y` in the coordinate system associated with the bottom right corner of the kernel.

The output value is computed as a sum of neighbor pixels' values, with kernel matrix elements used as weight factors. Note that summation formulas implement a convolution operation, which means that kernel coefficients are used in inverse order. Optionally, the output pixel values may be scaled. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

---

## Filter

*Filters an image using a general rectangular kernel.*

---

### Syntax

#### Case 1: Operation on integer data.

```
IppStatus ippiFilter_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32s*
    pKernel, IppiSize kernelSize, IppiPoint anchor, int divisor);
```

Supported values for *mod*:

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R
8u_AC4R	16s_AC4R

### Case 2: Operation on floating-point data.

```
IppStatus ippiFilter_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    IppiSize kernelSize, IppiPoint anchor);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the kernel values.
<i>kernelSize</i>	Size of the rectangular kernel in pixels.
<i>anchor</i>	Anchor cell specifying the rectangular kernel alignment with respect to the position of the input pixel.
<i>divisor</i>	The integer value by which the computed result is divided (for operations on integer data only).

### Description

The function `ippiFilter` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function uses the general rectangular kernel of size *kernelSize* to filter an image ROI. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *anchor*. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the bottom right corner of the kernel.

Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel.



In case of integer data, the result is divided by the fixed scaling factor *divisor*. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

For function flavors that operate on integer data, the result value  $Y_{i,j}$  for pixel  $X_{i,j}$  inside image's ROI is computed according to the following formula:

$$Y_{i,j} = \frac{1}{divisor} \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} X_{i+n-Q, j+m-P} \times K_{W-n-1, H-m-1}$$

where

$K_{n,m}$  are the kernel values;

$H = kernelSize.height$  is the vertical size of the kernel;

$W = kernelSize.width$  is the horizontal size of the kernel;

$P = H - anchor.y - 1$

$Q = W - anchor.x - 1$

Function flavors that accept input data of `Ipp32f` type use the same summation formula, but no scaling of the result is done.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pKernel</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> or <i>kernelSize</i> has a field with zero or negative value.
<code>ippStsDivisorErr</code>	Indicates an error condition if the <i>divisor</i> value is zero.

## Filter32f

*Filters an image with integer data using a floating-point rectangular kernel.*

---

### Syntax

```
IppStatus ippiFilter32f_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32f* pKernel, IppiSize kernelSize, IppiPoint anchor);
```

Supported values for *mod*:

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R
8u_AC4R	16s_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the kernel values.
<i>kernelSize</i>	Size of the rectangular kernel in pixels.
<i>anchor</i>	Anchor cell specifying the rectangular kernel alignment with respect to the position of the input pixel.

### Description

The function `ippiFilter32f` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function uses the rectangular kernel of floating-point values to filter an image that consists of integer data. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *anchor*. The anchor cell is specified by its coordinates *anchor.x* and

*anchor.y* in the coordinate system associated with the bottom right corner of the kernel. Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel. The summation formula for computing result values is similar to that used by `ippiFilter` function, but no scaling of the result is done. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pKernel</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> or <i>kernelSize</i> has a field with zero or negative value.

## Separable Filters

Separable filters use a spatial kernel consisting of a single column (as in `FilterColumn` function) or a single row (as in `FilterRow` function) to filter the source image.

---

### FilterColumn

*Filters an image using a spatial kernel that consists of a single column.*

---

#### Syntax

##### Case 1: Operation on integer data.

```
IppStatus ippiFilterColumn_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32s* pKernel, int kernelSize, int yAnchor, int divisor);
```

Supported values for *mod*:

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R
8u_AC4R	16s_AC4R

##### Case 2: Operation on floating-point data.

```
IppStatus ippiFilterColumn_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel,
    int kernelSize, int yAnchor);
```

Supported values for *mod*:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the kernel values.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>yAnchor</i>	Anchor cell specifying the kernel vertical alignment with respect to the position of the input pixel.
<i>divisor</i>	The integer value by which the computed result is divided (for operations on integer data only).

## Description

The function `ippiFilterColumn` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function uses the vertical column kernel of size *kernelSize* to filter an image ROI. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *yAnchor*. Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel. In case of integer data, the result is divided by the fixed scaling factor *divisor*. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

For function flavors that operate on integer data, the result value  $Y_{i,j}$  for pixel  $X_{i,j}$  inside image's ROI is computed according to the following formula:

$$Y_{i,j} = \frac{1}{divisor} \sum_{m=0}^{H-1} X_{i,j+m-P} \times K_{H-m-1}$$

where:

$K_m$  are the kernel values;

$H = kernelSize$  is the size of the vertical column kernel;

$P = H - yAnchor - 1$

Function flavors that accept input data of `Ipp32f` type use the same summation formula, but no scaling of the result is done.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with zero or negative value.
<code>ippStsDivisorErr</code>	Indicates an error condition if the <i>divisor</i> value is zero.

---

## FilterColumn32f

*Filters an image with integer data using a floating-point column kernel.*

---

### Syntax

```
IppStatus ippFilterColumn32f_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32f* pKernel, int kernelSize, int yAnchor);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16s_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>
<code>8u_C4R</code>	<code>16s_C4R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.

---

<i>pKernel</i>	Pointer to the column kernel values.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>yAnchor</i>	Anchor cell specifying the kernel vertical alignment with respect to the position of the input pixel.

## Description

The function `ippiFilterColumn32f` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function uses the vertical column kernel of floating-point values to filter an image that consists of integer data. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *yAnchor*. Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel. The summation formula for computing result values is similar to that used by `ippiFilterColumn` function, but no scaling of the result is done. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.

---

## FilterRow

*Filters an image using a spatial kernel that consists of a single row.*

---

## Syntax

### Case 1: Operation on integer data.

```
IppStatus ippiFilterRow_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32s* pKernel, int kernelSize, int xAnchor, int divisor);
```

Supported values for *mod*:

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R
8u_AC4R	16s_AC4R

### Case 2: Operation on floating-point data.

```
ippStatus ippiFilterRow_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize dstRoiSize, const Ipp32f* pKernel, int kernelSize,
    int xAnchor);
```

Supported values for *mod*:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the kernel values.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>xAnchor</i>	Anchor cell specifying the kernel horizontal alignment with respect to the position of the input pixel.
<i>divisor</i>	The integer value by which the computed result is divided (for operations on integer data only).

### Description

The function `ippiFilterRow` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).



This function uses the horizontal row kernel of size *kernelSize* to filter an image. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *xAnchor*. Note that kernel coefficients are used in inverse order.

The sum is written to the destination pixel. In case of integer data, the result is divided by the fixed scaling factor *divisor*. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

For function flavors that operate on integer data, the result value  $Y_{i,j}$  for pixel  $X_{i,j}$  inside image's ROI is computed according to the following formula:

$$Y_{i,j} = \frac{1}{divisor} \sum_{n=0}^{W-1} X_{i+n-Q,j} \times K_{W-n-1}$$

where

$K_n$  are the kernel values;

$W = kernelSize$  is the size of the horizontal row kernel;

$Q = W - xAnchor - 1$

Function flavors that accept input data of `Ipp32f` type use the same summation formula, but no scaling of the result is done.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pKernel</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsDivisorErr</code>	Indicates an error condition if the <i>divisor</i> value is zero.

## FilterRow32f

*Filters an image with integer data using a floating-point row kernel.*

---

### Syntax

```
IppStatus ippiFilterRow32f_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    const Ipp32f* pKernel, int kernelSize, int xAnchor);
```

Supported values for *mod*:

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_C4R	16s_C4R
8u_AC4R	16s_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the row kernel values.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>xAnchor</i>	Anchor cell specifying the kernel horizontal alignment with respect to the position of the input pixel.

### Description

The function `ippiFilterRow32f` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function uses the horizontal row kernel of floating-point values to filter an image that consists of integer data. This function sums the products between the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *xAnchor*. Note that kernel coefficients are used in inverse order. The sum is written to the destination pixel. The summation formula for computing result values is similar to that used by `ippiFilterRow` function, but no scaling of the result is done. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.

---

## FilterRowBorderPipelineGetBufferSize

*Computes the size of working buffer for the row filter.*

---

### Syntax

```
IppStatus ippiFilterRowBorderPipelineGetBufferSize_<mod>(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
```

Supported values for *mod*:

<code>8u16s_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
<code>8u16s_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>pBufferSize</i>	Pointer to the computed size of the buffer.

## Description

The function `ippiFilterRowBorderPipelineGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the size of the working buffer required for `ippiFilterRowBorderPipeline` function. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than `roiSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <code>pBufferSize</code> is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskSize</code> has a field with zero or negative value, or if <code>roiWidth</code> is less than 1.

---

## FilterRowBorderPipelineGetBufferSize\_Low

*Compute the size of working buffer for the Low-flavor of row filters.*

---

## Syntax

```
IppStatus ippiFilterRowBorderPipelineGetBufferSize_Low_<mod>(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
```

Supported values for *mod*:

```
16s_C1R
16s_C3R
```

## Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>pBufferSize</i>	Pointer to the computed size of the buffer.

## Description

The function `ippiFilterRowBorderPipelineGetBufferSize_Low` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the size of the working buffer required for the function `ippiFilterRowBorderPipeline_Low`. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than `roiSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <code>pBufferSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskSize</code> has a field with zero or negative value, or if <code>roiWidth</code> is less than 1.

---

## FilterColumnPipelineGetBufferSize

*Compute the size of working buffer for the column filter.*

---

## Syntax

```
IppStatus ippiFilterColumnPipelineGetBufferSize_<mod>(IppiSize roiSize,  
int kernelSize, int* pBufferSize);
```

Supported values for `mod`:

<code>8u16s_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
<code>8u16s_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>

## Parameters

<code>roiSize</code>	Maximum size of the source and destination image ROI.
<code>kernelSize</code>	Size of the kernel in pixels.
<code>pBufferSize</code>	Pointer to the computed size of the buffer.

## Description

The function `ippiFilterColumnPipelineGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the size of the working buffer required for `ippiFilterColumnPipeline` function. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than `roiSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <code>pBufferSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskSize</code> has a field with zero or negative value, or if <code>roiWidth</code> is less than 1.

---

## FilterColumnPipelineGetBufferSize\_Low

*Compute the size of working buffer for the Low-flavor of column filters.*

---

## Syntax

```
IppStatus ippiFilterColumnPipelineGetBufferSize_Low_<mod>(IppiSize
    roiSize, int kernelSize, int* pBufferSize);
```

Supported values for *mod*:

```
16s_C1R
16s_C3R
```

## Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>pBufferSize</i>	Pointer to the computed size of the buffer.

## Description

The function `ippiFilterColumnPipelineGetBufferSize_Low` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the size of the working buffer required for `ippiFilterColumnPipeline_Low` function. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than `roiSize`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <code>pBufferSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>maskSize</code> has a field with zero or negative value, or if <code>roiWidth</code> is less than 1.

---

## FilterRowBorderPipeline

*Applies the filter with border to image rows.*

---

## Syntax

### Case 1: Operation on one-channel integer data.

```
ippStatus ippiFilterRowBorderPipeline_<mod>(const Ipp<srcDatatype>*
    pSrc, int srcStep, Ipp<dstDatatype>** ppDst, IppiSize roiSize, const
    Ipp<dstDatatype>* pKernel, int kernelSize, int xAnchor,
    IppiBorderType borderType, Ipp<srcDatatype> borderValue, int divisor,
    Ipp8u* pBuffer);
```

Supported values for *mod*:

`8u16s_C1R`      `16s_C1R`

### Case 2: Operation on one-channel floating point data.

```
ippStatus ippiFilterRowBorderPipeline_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f** ppDst, IppiSize roiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor, IppiBorderType borderType, Ipp32f
    borderValue, Ipp8u* pBuffer);
```

### Case 3: Operation on three-channel integer data.

```
IppStatus ippiFilterRowBorderPipeline_<mod>(const Ipp<srcDatatype>*
    pSrc, int srcStep, Ipp<dstDatatype>** ppDst, IppiSize roiSize, const
    Ipp<dstDatatype>* pKernel, int kernelSize, int xAnchor,
    IppiBorderType borderType, Ipp<srcDatatype> borderValue[3], int
    divisor, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u16s\_C3R      16s\_C3R      32f\_C3R

### Case 4: Operation on three-channel floating point data.

```
IppStatus ippiFilterRowBorderPipeline_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f** ppDst, IppiSize roiSize, const Ipp32f* pKernel, int
    kernelSize, int xAnchor, IppiBorderType borderType, Ipp32f
    borderValue[3], Ipp8u* pBuffer);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.												
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.												
<i>ppDst</i>	Double pointer to the destination image ROI.												
<i>roiSize</i>	Size of the source and destination ROI in pixels.												
<i>pKernel</i>	Pointer to the row kernel values.												
<i>kernelSize</i>	Size of the kernel in pixels.												
<i>xAnchor</i>	Anchor value specifying the kernel row alignment with respect to the position of the input pixel.												
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table> <tr> <td><code>ippBorderZero</code></td><td>Values of all border pixel are set to zero.</td></tr> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderZero</code>	Values of all border pixel are set to zero.	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderZero</code>	Values of all border pixel are set to zero.												
<code>ippBorderConst</code>	Values of all border pixels are set to constant.												
<code>ippBorderRepl</code>	Replicated border is used.												
<code>ippBorderWrap</code>	Wrapped border is used												
<code>ippBorderMirror</code>	Mirrored border is used												
<code>ippBorderMirrorR</code>	Mirrored border with replication is used												
<i>borderValue</i>	The constant value (constant vector in case of three-channel data) to assign to the pixels in the constant border (not applicable for other border's type).												



<i>divisor</i>	Value by which the computed result is divided (for operations on integer data only).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

The function `ippiFilterRowBorderPipeline` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the horizontal row filter of the separable convolution kernel to the source image *pSrc*. The filter coefficients are placed in the reversed order. For integer data:

$$ppDst[i][j] = \frac{1}{divisor} \cdot \sum_{k=0}^{kernelSize-1} pSrc[i, j+k-xAnchor] \cdot pKernel[k]$$

and for floating point data:

$$ppDst[i][j] = \sum_{k=0}^{kernelSize-1} pSrc[i, j+k-xAnchor] \cdot pKernel[k]$$

Here  $j = 0, \dots, roiSize.width-1, i=0, \dots, roiSize.height-1$ . The values of pixels of the source image that lies outside of the image ROI (that is, if for pixel  $pSrc[i, l]$   $l \notin [0, roiSize.width-1]$  are set in accordance with the specified parameters *borderType* and *borderValue*.

This function can be used to organize the separable convolution as a step of the image processing pipeline (see [Example 9-3](#)).

The function requires the external buffer *pBuffer*, its size should be previously computed by the function [ippiFilterRowBorderPipelineGetBufferSize](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>xAnchor</i> has a wrong value.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has a wrong value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>divisor</i> is equal to 0.

---

## FilterRowBorderPipeline\_Low

*Applies the Low-flavor filter with border to image rows.*

---

### Syntax

#### Case 1: Operation on one-channel data.

```
IppStatus ippFilterRowBorderPipeline_Low_16s_C1R(const Ipp16s* pSrc,
        int srcStep, Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel,
        int kernelSize, int xAnchor, IppiBorderType borderType, Ipp16s
        borderValue, int divisor, Ipp8u* pBuffer);
```

#### Case 2: Operation on three-channel data.

```
IppStatus ippFilterRowBorderPipeline_Low_16s_C3R(const Ipp16s* pSrc,
        int srcStep, Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel,
        int kernelSize, int xAnchor, IppiBorderType borderType, Ipp16s
        borderValue[3], int divisor, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>ppDst</i>	Double pointer to the destination image ROI.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>pKernel</i>	Pointer to the row kernel values.

<i>kernelSize</i>	Size of the kernel in pixels.
<i>xAnchor</i>	Anchor cell specifying the kernel row alignment with respect to the position of the input pixel.
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <i>ippBorderZero</i> Values of all border pixel are set to zero. <i>ippBorderConst</i> Values of all border pixels are set to constant. <i>ippBorderRepl</i> Replicated border is used. <i>ippBorderWrap</i> Wrapped border is used <i>ippBorderMirror</i> Mirrored border is used <i>ippBorderMirrorR</i> Mirrored border with replication is used
<i>borderValue</i>	The constant value (constant vector in case of three-channel data) to assign to the pixels in the constant border (not applicable for other border's type).
<i>divisor</i>	Value by which the computed result is divided.
<i>pBuffer</i>	Pointer to the working buffer.

## Description

The function `ippiFilterRowBorderPipeline_Low` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the horizontal row filter of the separable convolution kernel to the source image *pSrc*. The filter coefficients are placed in the reversed order. The following formula is used:

$$ppDst[i][j] = \frac{1}{divisor} \cdot \sum_{k=0}^{kernelSize-1} pSrc[i, j+k-xAnchor] \cdot pKernel[k]$$

Here  $j = 0, \dots, roiSize.width-1, i=0, \dots, roiSize.height-1$ . The values of pixels of the source image that lies outside of the image ROI (that is, if for pixel  $pSrc[i, l]$   $l \notin [0, roiSize.width-1]$  are set in accordance with the specified parameters *borderType* and *borderValue*.

The function `ippiFilterRowBorderPipeline_Low` supposes that sum in the formula can be represented by a 32-bit integer number.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
--------------------------	---

<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsAnchorErr</code>	Indicates an error condition if <i>xAnchor</i> has a wrong value.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has a wrong value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>divisor</i> is equal to 0.

---

## FilterColumnPipeline

*Applies the filter to image columns.*

---

### Syntax

#### Case 1: Operation on integer data.

```
IppStatus ippiFilterColumnPipeline_<mod>(const Ipp<srcDatatype>** ppSrc,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, const
    Ipp<dstDatatype>* pKernel, int kernelSize, int divisor, Ipp8u*
    pBuffer);
```

Supported values for *mod*:

```
16s8u_C1R    16s_C1R
16s8u_C3R    16s_C3R
```

#### Case 2: Operation on floating-point data.

```
IppStatus ippiFilterColumnPipeline_<mod>(const Ipp<srcDatatype>** ppSrc,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, const
    Ipp<dstDatatype>* pKernel, int kernelSize, Ipp8u* pBuffer);
```

Supported values for *mod*:

32f\_C1R

32f\_C3R

## Parameters

<i>ppSrc</i>	Double pointer to the source image ROI.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>pKernel</i>	Pointer to the row kernel values.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>divisor</i>	Value by which the computed result is divided (for operations on integer data only).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

The function `ippiFilterColumnPipeline` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the column filter of the separable convolution kernel to the source image *pSrc*. The filter coefficients are placed in the reversed order. For integer data:

$$pDst[i, j] = \frac{1}{divisor} \cdot \sum_{k=0}^{kernelSize-1} pSrc[i+k, j] \cdot pKernel[k]$$

and for floating point data:

$$pDst[i, j] = \sum_{k=0}^{kernelSize-1} pSrc[i+k, j] \cdot pKernel[k]$$

Here  $j = 0, \dots, dstRoiSize.width-1, i=0, \dots, dstRoiSize.height-1$ .

The size of the source image is

$$(dstRoiSize.height + kernelSize - 1) * dstRoiSize.width.$$

This function can be used to organize the separable convolution as a step of image processing pipeline (see [Example 9-3](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error condition if <code>divisor</code> is equal to 0.

---

## FilterColumnPipeline\_Low

*Applies the Low-flavor filter to image columns.*

---

### Syntax

```

IppStatus ippiFilterColumnPipeline_Low_16s_C1R(const Ipp16s** ppSrc,
        Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp16s*
        pKernel, int kernelSize, int divisor, Ipp8u* pBuffer, Ipp8u*
        pBuffer);

IppStatus ippiFilterColumnPipeline_Low_16s_C3R(const Ipp16s** ppSrc,
        Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, const Ipp16s*
        pKernel, int kernelSize, int divisor, Ipp8u* pBuffer, Ipp8u*
        pBuffer);

```

### Parameters

`ppSrc` Double pointer to the source image ROI.

<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>pKernel</i>	Pointer to the row kernel values.
<i>kernelSize</i>	Size of the kernel in pixels.
<i>divisor</i>	Value by which the computed result is divided (for operations on integer data only).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

The function `ippiFilterColumnPipeline_Low` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the column filter of the separable convolution kernel to the source image *pSrc*. The filter coefficients are placed in the reversed order. The following formula is used:

$$pDst[i, j] = \frac{1}{divisor} \cdot \sum_{k=0}^{kernelSize-1} pSrc[i+k, j] \cdot pKernel[k]$$

The function `ippiFilterColumnPipeline_Low` supposes that sum in the formula can be represented by a 32-bit integer number.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>divisor</i> is equal to 0.

### Example 9-3    **Separable Convolution 3x3 by One Row**

---

```
void Separable_3x3(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep,
                  IppiSize roiSize, Ipp16s* pKerX, Ipp16s* pKerY) {
    Ipp16s **get, *dst=pDst;
    const Ipp16s *src=pSrc;
    IppiSize roi;
    int todo=roiSize.height,sizeRow,sizeCol,bufLen;
    int mStep=(roiSize.width+7)&(~7),sStep=srcStep>>1,dStep=dstStep>>1;
    Ipp8u *pBufRow, *pBufCol;
    ippiFilterRowBorderLowPipelineGetBufferSize_16s_C1R(roiSize,3,&sizeRow);
    ippiFilterColumnLowPipelineGetBufferSize_16s_C1R(roiSize,3,&sizeCol);
    bufLen = mStep*3*sizeof(Ipp16s)+4*sizeof(Ipp16s*);
    pBufRow = ippsMalloc_8u(sizeRow);
    pBufCol = ippsMalloc_8u(sizeCol);
    get = (Ipp16s**)ippsMalloc_8u(bufLen);
    get[0]=get[1]=(Ipp16s*)(get+4);
    get[2]=get[1]+mStep;
    get[3]=get[2]+mStep;
    roi.width  = roiSize.width;
    roi.height = 1;
    ippiFilterRowBorderLowPipeline_16s_C1R(src, srcStep, get, roi, pKerX,
        3, 1, ippBorderRepl, 0, 1, pBufRow);
    if (--todo) {
        get[2] = get[0];
    } else {
        get[2] = get[0] + mStep; get[3] = get[2] + mStep;
        for (; todo>0; src+=sStep, dst+=dStep, todo--) {
            ippiFilterRowBorderLowPipeline_16s_C1R(src, srcStep, get+2, roi, pKerX,
                3, 1, ippBorderRepl, 0, 1, pBufRow);
            ippiFilterColumnLowPipeline_16s_C1R(get, dst, dstStep, roi, pKerY,
                3, 1, pBufCol);
            get[0] = get[1]; get[1] = get[2]; get[2] = get[3]; get[3] = get[0];
        }
    }
    ippiFilterColumnLowPipeline_16s_C1R(get, dst, dstStep, roi, pKerY,
        3, 1, pBufCol);

    ippsFree(pBufRow);
    ippsFree(pBufCol);
    ippsFree(get);
}
```

---



## Wiener Filters

Intel IPP functions described in this section perform adaptive noise-removal filtering of an image using Wiener filter [Lim90]. The adaptive filter is more selective than a comparable linear filter in preserving edges and other high frequency parts of an image. Wiener filters are commonly used in image processing applications to remove additive noise from degraded images, to restore a blurry image, and in similar operations.

These functions use a pixel-wise adaptive Wiener method based on statistics estimated from a local neighborhood (mask) of arbitrary size for each pixel.

---

### FilterWienerGetBufferSize

*Computes the size of the external buffer for `ippiFilterWiener` function.*

---

#### Syntax

```
IppStatus ippiFilterWienerGetBufferSize(IppiSize dstRoiSize,
    IppiSize maskSize, int channels, int* pBufferSize);
```

#### Parameters

<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>channels</i>	Number of channels in the image.
<i>pBufferSize</i>	Pointer to the computed value of the external buffer size.

#### Description

The function `ippiFilterWienerGetBufferSize` is declared in the `ippi.h` file. This function computes the size in bytes of an external memory buffer that is required for the function [ippiFilterWiener](#), and stores the result in the *pBufferSize*.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is NULL.

<code>ippStsSizeErr</code>	Indicates an error condition if one of the fields of <code>dstRoiSize</code> has zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if one of the fields of <code>maskSize</code> has zero or negative value.
<code>ippStsNumChannelsErr</code>	Indicates an error condition if <code>channels</code> is not 1, 3 or 4.

---

## FilterWiener

*Filters an image using the Wiener algorithm.*

---

### Syntax

#### Case 1: Operation on one-channel images.

```
IppStatus ippFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiSize maskSize, IppiPoint anchor, Ipp32f noise[1], Ipp8u*
    pBuffer);
```

Supported values for `mod` :

`8u_C1R`            `16s_C1R`            `32f_C1R`

#### Case 2: Operation on multi-channel images.

```
IppStatus ippFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize
    maskSize, IppiPoint anchor, Ipp32f noise[3], Ipp8u* pBuffer);
```

Supported values for `mod` :

`8u_C3R`            `16s_C3R`            `32f_C3R`  
`8u_AC4R`           `16s_AC4R`           `32f_AC4R`

```
IppStatus ippFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize
    maskSize, IppiPoint anchor, Ipp32f noise[4], Ipp8u* pBuffer);
```

Supported values for `mod` :

`8u_C4R`            `16s_C4R`            `32f_C4R`

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>maskSize</i>	Size of the mask in pixels.
<i>anchor</i>	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
<i>noise</i>	Noise level value or array of the noise level values in case of multi-channel image. This value must be in the range [0,1].
<i>pBuffer</i>	Pointer to the external work buffer.

## Description

The function `ippiFilterWiener` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function performs adaptive filtering of the image degraded by constant power additive noise. For each pixel of the input image *pSrc*, the function estimates the local image mean  $\mu$  and variance  $\sigma$  in the rectangular neighborhood (mask) of size *maskSize* with anchor cell *anchor* centered on the pixel. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the bottom right corner of the mask.

The following formulas are used in computations:

$$\mu_{i,j} = \frac{1}{HW} \cdot \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} x_{m,n}$$

$$\sigma_{i,j}^2 = \frac{1}{HW} \cdot \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} x_{m,n}^2 - \mu_{i,j}^2$$

Here  $\mu_{i,j}$  and  $\sigma_{i,j}$  stand for local mean and variance for pixel  $X_{i,j}$ , respectively, and  $H, W$  are the vertical and horizontal sizes of the mask, respectively.

The corresponding value for the output pixel  $Y_{i,j}$  is computed as:

$$Y_{i,j} = \mu_{i,j} + \frac{\sigma_{i,j}^2 - v^2}{\sigma^2} \cdot [X_{i,j} - \mu_{i,j}]$$

and stored in the *pDst*. Here  $v^2$  is the noise variance, specified for each channel by the noise level parameter *noise*. If this parameter is not defined (*noise* = 0), then the function estimates the noise level by averaging through the image of all local variances  $\sigma_{i,j}$ , and stores the corresponding values in the *noise* for further use.

The function `ippiFilterWiener` uses the external work buffer *pBuffer*, which must be allocated before the function call. To determine the required buffer size, the function [`ippiFilterWienerGetBufferSize`](#) can be used.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if one of the fields of <i>dstRoiSize</i> has zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if one of the fields of <i>maskSize</i> has zero or negative value.
<code>ippStsNoiseRangeErr</code>	Indicates an error condition if one of the <i>noise</i> values is less than 0 or greater than 1.

## Convolution

Intel IPP functions described in this section perform two-dimensional finite linear convolution operation between two source images and write the result into the destination image. Convolution is used to perform many common image processing operations including sharpening, blurring, noise reduction, embossing, and edge enhancement.

For convenience, in this section we shall represent any digital image  $f$  as a matrix with  $M_f$  columns and  $N_f$  rows that contains pixel values  $f[i,j]$ ,  $0 \leq i < M_f$ ,  $0 \leq j < N_f$ .

---

## ConvFull

*Performs a full convolution of two images.*

---

### Syntax

#### Case 1: Operation on integer data.

```
IppStatus ippiConvFull_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize src2Size, Ipp<datatype>* pDst, int dstStep, int divisor);
```

Supported values for *mod*:

8u_C1R	16s_C1R
8u_C3R	16s_C3R
8u_AC4R	16s_AC4R

#### Case 2: Operation on floating-point data.

```
IppStatus ippiConvFull_<mod>(const Ipp32f* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp32f* pSrc2, int src2Step,
    IppiSize src2Size, Ipp32f* pDst, int dstStep);
```

Supported values for *mod*:

32f_C1R
32f_C3R
32f_AC4R

### Parameters

*pSrc1*, *pSrc2*      Pointers to the source images ROI.

<i>src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source images.
<i>src1Size, src2Size</i>	Sizes in pixels of the source images.
<i>pDst</i>	Pointer to the destination buffer ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>divisor</i>	The integer value by which the computed result is divided (for operations on integer data only).

### Description

The function `ippiConvFull` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function performs full two-dimensional finite linear convolution operation between two source images pointed to by *pSrc1* and *pSrc2*. If we denote the first source image as a matrix *f* of size *M<sub>f</sub>* by *N<sub>f</sub>* and the second source image as a matrix *g* of size *M<sub>g</sub>* by *N<sub>g</sub>*, then the destination image *h* obtained as a result of function operation will have size *M<sub>h</sub>* by *N<sub>h</sub>*, where *M<sub>h</sub>* = *M<sub>f</sub>* + *M<sub>g</sub>* - 1 and *N<sub>h</sub>* = *N<sub>f</sub>* + *N<sub>g</sub>* - 1.

The function `ippiConvFull` implements the following equation to compute values *h* [*i,j*] of the destination image:

$$h[i, j] = \frac{1}{divisor} \sum_{l=0}^{N_h-1} \sum_{k=0}^{M_h-1} f[k, l] \times g[i-k, j-l] \quad ,$$

where  $0 \leq i < M_h$ ,  $0 \leq j < N_h$  and

$$f[k, l] = \begin{cases} f[k, l], & 0 \leq k < M_f; \quad 0 \leq l < N_f \\ 0 & , otherwise \end{cases}$$

$$g[i-k, j-l] = \begin{cases} g[i-k, j-l], & 0 \leq i-k < M_g; \quad 0 \leq j-l < N_g \\ 0 & , otherwise \end{cases}$$

Function flavors that accept input data of `Ipp32f` type use the same summation formula, but no scaling of the result is done (*divisor* = 1 is assumed).

To illustrate the function operation, for the source images *f*, *g* of size 3 x 5 represented as

$$f = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad g = f$$

the resulting convolution image *h* is of size 5 x 9 and contains the following data:

$$h = \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 2 & 2 & 0 & 0 \\ 3 & 4 & 6 & 4 & 2 \\ 2 & 2 & 4 & 2 & 2 \\ 3 & 6 & 11 & 6 & 3 \\ 2 & 2 & 4 & 2 & 2 \\ 2 & 4 & 6 & 4 & 3 \\ 0 & 0 & 2 & 2 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc1</i> , <i>pSrc2</i> , or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>src1Size</i> or <i>src2Size</i> has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>src1Step</i> , <i>src2Step</i> , or <i>dstStep</i> has zero or negative value.

<code>ippStsDivisorErr</code>	Indicates an error condition if <i>divisor</i> has zero value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation failed.

---

## ConvValid

*Performs a valid convolution of two images.*

---

### Syntax

#### Case 1: Operation on integer data.

```
IppStatus ippConvValid_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize src2Size, Ipp<datatype>* pDst, int dstStep, int divisor);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>16s_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>

#### Case 2: Operation on floating-point data.

```
IppStatus ippConvValid_<mod>(const Ipp32f* pSrc1, int src1Step,
    IppiSize src1Size, const Ipp32f* pSrc2, int src2Step,
    IppiSize src2Size, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* :

<code>32f_C1R</code>
<code>32f_C3R</code>
<code>32f_AC4R</code>

### Parameters

<code>pSrc1, pSrc2</code>	Pointers to the source images ROI.
<code>src1Step, src2Step</code>	Distances in bytes between starts of consecutive lines in the source images.
<code>src1Size, src2Size</code>	Sizes in pixels of the source images.



<i>pDst</i>	Pointer to the destination buffer ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>divisor</i>	The integer value by which the computed result is divided (for operations on integer data only).

## Description

The function `ippiConvValid` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function performs valid two-dimensional finite linear convolution operation between two source images pointed to by *pSrc1* and *pSrc2*.

If we denote the first source image as a matrix  $f$  of size  $M_f$  by  $N_f$  and the second source image as a matrix  $g$  of size  $M_g$  by  $N_g$ , then the destination image  $h$  obtained as a result of function operation will have size  $M_h$  by  $N_h$ , where  $M_h = |M_f - M_g| + 1$  and  $N_h = |N_f - N_g| + 1$ .

The function `ippiConvFull` implements the following equation to compute values  $h[i, j]$  of the destination image:

$$h[i, j] = \frac{1}{divisor} \sum_{l=0}^{N_g-1} \sum_{k=0}^{M_g-1} f[i+k, j+l] \times g[M_g-k-1, N_g-l-1]$$

where  $0 \leq i < M_h$ ,  $0 \leq j < N_h$ .

We assume here that  $M_f \geq M_g$  and  $N_f \geq N_g$ . In case when  $M_f < M_g$  and  $N_f < N_g$ , the subscript index  $g$  in this equation must be replaced with index  $f$ . For any other combination of source image sizes, the function `ippiConvValid` performs no operation.

Note that the above formula gives the same result as in the case of `ippiConvFull` function, but produces only that part of the convolution image which is computed without using zero-padded values.

Function flavors that accept input data of `Ipp32f` type use the same summation formula, but no scaling of the result is done (*divisor* = 1 is assumed).

To illustrate the function operation, for the source images  $f$ ,  $g$  of size 3 x 5 represented as

$$f = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad g = f,$$

the resulting convolution image  $h$  is of size 1 x 1 and contains the following data:

$$h = \begin{bmatrix} 11 \end{bmatrix}.$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc1</i> , <i>pSrc2</i> , or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>src1Size</i> or <i>src2Size</i> has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>src1Step</i> , <i>src2Step</i> , or <i>dstStep</i> has zero or negative value.
<code>ippStsDivisorErr</code>	Indicates an error condition if <i>divisor</i> has zero value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation failed.

## Deconvolution

Functions described in this section perform image deconvolution. They can be used for restoring the degraded image, in particular image that was obtained by applying the convolution operation with known kernel. The Intel IPP functions implements two methods: Fourier deconvolution (noniterative method) [see for example, [Puetter2005](#)], and Richardson-Lucy method (iterative method) [[Richardson72](#)]. Border pixels of a source image are restored before deconvolution.

---

### DeconvFFTInitAlloc

*Allocates and initializes state structure for FFT deconvolution.*

---

#### Syntax

```
IppStatus ippiDeconvFFTInitAlloc_32f_C1R(IppiDeconvFFTState_32f_C1R**
    ppDeconvFFTState, const Ipp32f* pKernel, int kernelSize, int
    FFTOrder, Ipp32f threshold);

IppStatus ippiDeconvFFTInitAlloc_32f_C3R(IppiDeconvFFTState_32f_C3R**
    ppDeconvFFTState, const Ipp32f* pKernel, int kernelSize, int
    FFTOrder, Ipp32f threshold);
```

#### Parameters

<i>ppDeconvFFTState</i>	Double pointer to the FFT deconvolution state structure.
<i>pKernel</i>	Pointer to the kernel array.
<i>kernelSize</i>	Size of the kernel.
<i>FFTOrder</i>	Order of the created FFT state structure.
<i>threshold</i>	Value of the threshold level (to except dividing by zero).

#### Description

The function `ippiDeconvFFTInitAlloc` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory, initializes the deconvolution state structure and returns the pointer `ppDeconvFFTState` to it. This structure is used by the function [ippiDeconvFFT](#) that performs deconvolution of the source image using FFT.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>kernelSize</i> is less than or equal to 0, or if <i>kernelSize</i> is greater than $2^{FFTorder}$ .
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>threshold</i> is less than or equal to 0.

---

## DeconvFFTFree

*Frees memory allocated for the FFT deconvolution state structure.*

---

### Syntax

```
IppStatus ippideconvFFTFree_32f_C1R(IppiDeconvFFTState_32f_C1R*
    pDeconvFFTState)
IppStatus ippideconvFFTFree_32f_C3R(IppiDeconvFFTState_32f_C3R*
    pDeconvFFTState);
```

### Parameters

*pDeconvFFTState*      Pointer to the FFT deconvolution state structure.

### Description

The function `ippideconvFFTFree` is declared in the `ippi.h` file. This function frees memory allocated by the function [ippideconvFFTInitAlloc](#) for the FFT deconvolution state structure *pDeconvFFTState*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pDeconvFFTState</i> is NULL.

---

## DeconvFFT

*Performs FFT deconvolution of an image.*

---

### Syntax

```
IppStatus ippiDeconvFFT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, IppiDeconvFFTState_32f_C1R*
    pDeconvFFTState);

IppStatus ippiDeconvFFT_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, IppiDeconvFFTState_32f_C3R*
    pDeconvFFTState);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>pDeconvFFTState</i>	Pointer to the FFT deconvolution state structure.

### Description

The function `ippiDeconvFFT` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs deconvolution of the source image *pSrc* using FFT with parameters specified in the FFT deconvolution state structure *pDeconvFFTState* and stores results to the destination image *pDst*. The FFT deconvolution state structure must be initialized by calling the function [ippiDeconvFFTInitAlloc](#) beforehand.

---

## DeconvLRInitAlloc

*Allocates and initializes state structure for LR deconvolution.*

---

### Syntax

```
IppStatus ippiDeconvLRInitAlloc_32f_C1R(IppiDeconvLR_32f_C1R**  
    ppDeconvLR, const Ipp32f* pKernel, int kernelSize, IppiSize maxRoi,  
    Ipp32f threshold);  
  
IppStatus ippiDeconvLRInitAlloc_32f_C3R(IppiDeconvLR_32f_C3R**  
    ppDeconvLR, const Ipp32f* pKernel, int kernelSize, IppiSize maxRoi,  
    Ipp32f threshold);
```

### Parameters

<i>ppDeconvLR</i>	Double pointer to the LR deconvolution state structure.
<i>pKernel</i>	Pointer to the kernel array.
<i>kernelSize</i>	Size of the kernel.
<i>maxRoi</i>	Maximum size of the image ROI in pixels.
<i>threshold</i>	Value of the threshold level (to except dividing by zero).

### Description

The function `ippiDeconvLRInitAlloc` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory, initializes the deconvolution state structure and returns the pointer `ppDeconvLRState` to it. This structure is used by the function [ippiDeconvFFT](#) that performs deconvolution of the source image using Lucy-Richardson algorithm.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>kernelSize</i> is less than or equal to 0, or if <i>kernelSize</i> is greater than <i>maxRoi.width</i> or <i>maxRoi.height</i> , or if any field of the <i>maxRoi</i> is less than or equal to 0.

---

<code>ippStsBadArgErr</code>	Indicates an error condition if <i>threshold</i> is less than or equal to 0.
------------------------------	--

---

## DeconvLRFree

*Frees memory allocated for the LR deconvolution state structure.*

---

### Syntax

```
IppStatus ippDeconvLRFree_32f_C1R(IppiDeconvLR_32f_C1R* pDeconvLR)
IppStatus ippDeconvLRFree_32f_C3R(IppiDeconvLR_32f_C3R* pDeconvLR);
```

### Parameters

<i>pDeconvLR</i>	Pointer to the LR deconvolution state structure.
------------------	--

### Description

The function `ippDeconvLRFree` is declared in the `ippi.h` file. This function frees memory allocated by the function [ippiDeconvLRInitAlloc](#) for the LR deconvolution state structure *pDeconvLR*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pDeconvLR</i> is <code>NULL</code> .

---

## DeconvLR

*Performs LR deconvolution of an image.*

---

### Syntax

```
IppStatus ippDeconvLR_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, int numIter,
    IppiDeconvLR_32f_C1R* pDeconvLR);
```

```
IppStatus ippiDeconvLR_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, int numIter,
    IppiDeconvLR_32f_C3R* pDeconvLR);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>numIter</i>	Number of algorithm iterations.
<i>pDeconvLR</i>	Pointer to the LR deconvolution state structure.



## Fixed Filters

The fixed filter functions perform linear filtering of a source image using one of the predefined convolution kernels. The supported fixed filters and their respective kernel sizes are listed in the following table:

**Table 9-3**      **Types of the Fixed Filter Functions**

Fixed Filter Type	Kernel Size
Horizontal Prewitt operator	3x3
Vertical Prewitt operator	3x3
Horizontal Scharr operator	3x3
Vertical Scharr operator	3x3
Horizontal Sobel operator	3x3 or 5x5
Vertical Sobel operator	3x3 or 5x5
Second derivative horizontal Sobel operator	3x3 or 5x5
Second derivative vertical Sobel operator	3x3 or 5x5
Second cross derivative Sobel operator	3x3 or 5x5
Horizontal Roberts operator	3x3
Vertical Roberts operator	3x3
Laplacian highpass filter	3x3 or 5x5
Gaussian lowpass filter	3x3 or 5x5
Highpass filter	3x3 or 5x5
Lowpass filter	3x3 or 5x5
Sharpening filter	3x3

Using fixed filter functions with predefined kernels is more efficient as it eliminates the need to create the convolution kernel in your application program.



**NOTE.** For all fixed filter functions, to ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).




---

**NOTE.** The anchor cell is the center cell of the kernel for all fixed filters.

---



---

## FilterPrewittHoriz

*Filters an image using a horizontal Prewitt kernel.*

---

### Syntax

```
IppStatus ippiFilterPrewittHoriz_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiFilterPrewittHoriz` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a horizontal Prewitt operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```

1   1   1
0   0   0
-1  -1  -1

```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of enhancing horizontal edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

## FilterPrewittVert

*Filters an image using a vertical Prewitt kernel.*

---

### Syntax

```

IppStatus ippifilterPrewittVert_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);

```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiFilterPrewittVert` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a vertical Prewitt operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```
-1  0  1
-1  0  1
-1  0  1
```

The anchor cell is the center cell (red) of the kernel. The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of enhancing vertical edges of an image.

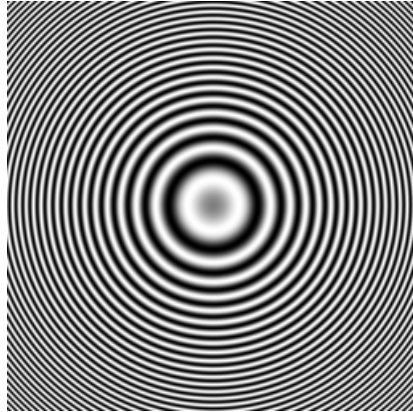
To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

[Figure 9-2](#) shows the result of filtering the sample image using the horizontal (upper part) and vertical (lower part) Prewitt operators. All pixels with negative values are zeroed because the function for `unsigned char` values was used.

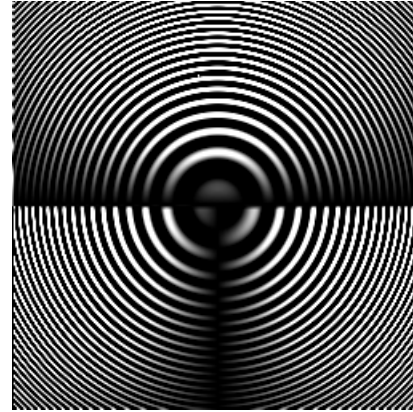
---

**Figure 9-2** Using Prewitt Operator for Image Filtering

---



Sample Image



Filtered Image

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## FilterScharrHoriz

*Filters an image using a horizontal Scharr kernel.*

---

### Syntax

```
IppStatus ippiFilterScharrHoriz_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

8u16s\_C1R      8s16s\_C1R      32f\_C1R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiFilterScharrHoriz` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a horizontal Scharr operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```

3   10   3
0   0   0
-3  -10  -3
```

The anchor cell is the center cell (red) of the kernel. The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of both enhancing and smoothing horizontal edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

## FilterScharrVert

*Filters an image using a vertical Scharr kernel.*

---

### Syntax

```
IppStatus ippFilterScharrVert_<mod>(const Ipp<srcDatatype>* pSrc,  
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

`8u16s_C1R`      `8s16s_C1R`      `32f_C1R`

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippFilterScharrVert` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a vertical Scharr operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```

3    0    -3
10   0   -10
3    0    -3

```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of both enhancing and smoothing vertical edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

## FilterSobelHoriz, FilterSobelHorizMask

*Filters an image using a horizontal Sobel kernel.*

---

### Syntax

```

IppStatus ippifilterSobelHoriz_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);

```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_C4R	16s_C4R	32f_C4R
8u_AC4R	16s_AC4R	32f_AC4R

```

IppStatus ippifilterSobelHoriz_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);

```



Supported values for *mod* :

8u16s\_C1R      8s16s\_C1R

```
IppStatus ippiFilterSobelHorizMask_32f_C1R(const Ipp32f* pSrc,
      int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize,
      IppiMaskSize mask);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <i>IppiMaskSize</i> type.

## Description

The functions `ippiFilterSobelHoriz` and `ippiFilterSobelHorizMask` are declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). These functions apply a horizontal Sobel operator to an image ROI. The appropriate kernel is the matrix of 3x3 size, or either 3x3 or 5x5 size in accordance with *mask* parameter of the corresponding function flavors. The kernels have the following values:

				1	4	6	4	1
				2	8	12	8	2
1	2	1		0	0	0	0	0
0	0	0	or	0	0	0	0	0
-1	-2	-1		-2	-8	-12	-8	-4
				-1	-4	-6	-4	-1

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of both enhancing and smoothing horizontal edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.

<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

---

## FilterSobelVert, FilterSobelVertMask

*Filter an image using a vertical Sobel kernel.*

---

### Syntax

```
IppStatus ippiFilterSobelVert_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>16s_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

```
IppStatus ippiFilterSobelVert_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

<code>8u16s_C1R</code>	<code>8s16s_C1R</code>
------------------------	------------------------

```
IppStatus ippiFilterSobelVertMask_32f_C1R(const Ipp32f* pSrc,
    int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

## Description

The functions `ippiFilterSobelVert` and `ippiFilterSobelVertMask` are declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). These functions apply a vertical Sobel operator to an image ROI. The appropriate kernel is the matrix of 3x3 size, or either 3x3 or 5x5 size in accordance with *mask* parameter of the corresponding function flavors. The kernels have the following values:

				-1	-2	0	2	1
-1	0	1		-4	-8	0	8	4
-2	0	2	or	-6	-12	0	12	6
-1	0	1		-4	-8	0	8	4
				-1	-2	0	2	1

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of both enhancing and smoothing vertical edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

# FilterSobelHorizSecond

Filters an image using a second derivative horizontal Sobel operator.

## Syntax

```
IppStatus ippiFilterSobelHorizSecond_<mod>(const Ipp<srcDatatype>* pSrc,
      int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
      IppiMaskSize mask);
```

Supported values for *mod* :

```
8u16s_C1R      8s16s_C1R      32f_C1R
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <i>IppiMaskSize</i> type.

## Description

The function `ippiFilterSobelHorizSecond` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a second derivative horizontal Sobel operator to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

				1	4	6	4	1
				0	0	0	0	0
1	2	1		-2	-8	-12	-8	-2
-2	-4	-2	or	0	0	0	0	0
1	2	1		1	4	6	4	1

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of both enhancing and smoothing horizontal edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

---

## FilterSobelVertSecond

*Filters an image using a second derivative vertical Sobel operator.*

---

### Syntax

```
IppStatus ippFilterSobelVertSecond_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

`8u16s_C1R`      `8s16s_C1R`      `32f_C1R`

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of IppiMaskSize type.

### Description

The function `ippiFilterSobelVertSecond` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a second derivative vertical Sobel operator to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

				1	0	-2	0	1
1	-2	1		4	0	-8	0	4
2	-4	2	or	6	0	-12	0	6
1	-2	1		4	0	-8	0	4
				1	0	-2	0	1

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI. This filter has the effect of both enhancing and smoothing vertical edges of an image. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

---

## FilterSobelCross

*Filters an image using a second cross derivative Sobel operator.*

---

### Syntax

```
IppStatus ippiFilterSobelCross_<mod>(const Ipp<srcDatatype>* pSrc,  
    int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,  
    IppiMaskSize mask);
```

Supported values for *mod* :

8u16s\_C1R      8s16s\_C1R      32f\_C1R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

### Description

The function `ippiFilterSobelCross` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a second cross derivative Sobel operator to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

				-1	-2	0	2	1
-1	0	1		-2	-4	0	4	2
0	0	0	or	0	0	0	0	0
1	0	-1		2	4	0	-4	-2
				1	2	0	-2	-1

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter has the effect of both enhancing and smoothing vertical edges of an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

---

## FilterRobertsDown

*Filters an image using a horizontal Roberts kernel.*

---

### Syntax

```
IppStatus ippiFilterRobertsDown_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.



---

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiFilterRobertsDown` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a horizontal Roberts operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```
0  0  0
0  1  0
0  0 -1
```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter gives the gross approximation of the pixel values' gradient in the horizontal direction.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

## FilterRobertsUp

*Filters an image using a vertical Roberts kernel.*

---

## Syntax

```
IppStatus ippiFilterRobertsUp_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiFilterRobertsUp` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a vertical Roberts operator to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

```

0  0  0
0  1  0
-1  0  0

```

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI. This filter gives the gross approximation of the pixel values' gradient in the vertical direction.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

# FilterLaplace

Filters an image using a Laplacian kernel.

## Syntax

```
IppStatusippiFilterLaplace_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R	8u16s_C1R	8s16s_C1R
8u_C3R	16s_C3R	32f_C3R		
8u_C4R	16s_C4R	32f_C4R		
8u_AC4R	16s_AC4R	32f_AC4R		

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of IppiMaskSize type.

## Description

The function `ippiFilterLaplace` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a highpass Laplacian filter to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

				-1	-3	-4	-3	-1
-1	-1	1		-3	0	6	0	-3
-1	8	1	or	-4	6	20	6	-4
-1	-1	1		-3	0	6	0	-3

-1 -3 -4 -3 -1

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.  
This filter helps locate zero crossings in an image.  
To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

---

**FilterGauss**

*Filters an image using a Gaussian kernel.*

---

**Syntax**

```
IppStatus ippiFilterGauss_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>16s_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
-------------	----------------------------------

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

## Description

The function `ippiFilterGauss` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a lowpass Gaussian filter to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size.

The 3x3 filter uses the kernel:

$$\begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

These filter coefficients correspond to a 2-dimensional Gaussian distribution with standard deviation 0.85.

The 5x5 filter uses the kernel:

$$\begin{bmatrix} 2/571 & 7/571 & 12/571 & 7/571 & 2/571 \\ 7/571 & 31/571 & 52/571 & 31/571 & 7/571 \\ 12/571 & 52/571 & 127/571 & 52/571 & 12/571 \\ 7/571 & 31/571 & 52/571 & 31/571 & 7/571 \\ 2/571 & 7/571 & 12/571 & 7/571 & 2/571 \end{bmatrix}$$

These filter coefficients correspond to a 2-dimensional Gaussian distribution with standard deviation 1.0.

The anchor cell is the center cell of the kernels (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <code>mask</code> has an illegal value.

---

## FilterHipass

*Filters an image using a highpass filter.*

---

### Syntax

```
IppStatus ippiFilterHipass_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for `mod` :

<code>8u_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>16s_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>dstRoiSize</code>	Size of the source and destination ROI in pixels.
<code>mask</code>	Predefined mask of <code>IppiMaskSize</code> type.

## Description

The function `ippiFilterHipass` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a highpass filter to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size with the following values:

		-1	-1	-1	-1	-1
-1	-1	-1		-1	-1	-1
-1	8	-1	or	-1	-1	24
-1	-1	-1		-1	-1	-1
				-1	-1	-1

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to `dstRoiSize`, the size of the destination image ROI.

This filter attenuates low-frequency components and thus sharpens an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <code>mask</code> has an illegal value.

---

## FilterLowpass

*Filters an image using a lowpass filter.*

---

## Syntax

```
IppStatus ippiFilterLowpass_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiMaskSize mask);
```

Supported values for *mod* :

8u_C1R	16s_C1R	32f_C1R
8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.

## Description

The function `ippiFilterLowpass` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a lowpass filter to an image ROI. The corresponding kernel is the matrix of either 3x3 or 5x5 size.

The 3x3 filter uses the kernel

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

The 5x5 filter uses the kernel

1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to *dstRoiSize*, the size of the destination image ROI.

This filter blurs an image by averaging the pixel values over some neighborhood.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has an illegal value.

---

## FilterSharpen

*Filters an image using a sharpening filter.*

---

### Syntax

```
IppStatus ippFilterSharpen_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>16s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16s_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>16s_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>16s_AC4R</code>	<code>32f_AC4R</code>

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiFilterSharpen` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies a sharpening filter to an image ROI. The corresponding kernel is the matrix of 3x3 size with the following values:

$$\begin{bmatrix} -1/8 & -1/8 & -1/8 \\ -1/8 & 16/8 & -1/8 \\ -1/8 & -1/8 & -1/8 \end{bmatrix}$$

The anchor cell is the center cell of the kernel (red). The size of the source image ROI is equal to `dstRoiSize`, the size of the destination image ROI. This filter enhances high-frequency components and thus sharpens an image.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see [Borders](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

## Fixed Filters with Border

This section describes the fixed filter functions that perform linear filtering of a source image using one of the predefined convolution kernels like the filter functions from the previous section do. Difference is that these functions automatically create a required border and define appropriate pixel values.

---

### FilterScharrHorizGetBufferSize

*Computes the size of the external buffer for the horizontal Scharr filter with border.*

---

#### Syntax

```
IppStatus ippiFilterScharrHorizGetBufferSize_8u16s_C1R(IppiSize roiSize,
    int* pBufferSize);
IppStatus ippiFilterScharrHorizGetBufferSize_32f_C1R(IppiSize roiSize,
    int* pBufferSize);
```

#### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>pBufferSize</i>	Pointer to the buffer size.

#### Description

The function `ippiFilterScharrHorizGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the function [ippiFilterScharrHorizBorder](#). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

## FilterScharrVertGetBufferSize

*Computes the size of the external buffer for the vertical Scharr filter with border.*

---

### Syntax

```
IppStatus ippiFilterScharrVertGetBufferSize_8u16s_C1R(IppiSize roiSize,
    int* pBufferSize);
IppStatus ippiFilterScharrVertGetBufferSize_32f_C1R(IppiSize roiSize,
    int* pBufferSize);
```

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>pBufferSize</i>	Pointer to the buffer size.

### Description

The function `ippiFilterScharrVertGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the function [ippiFilterScharrVertBorder](#). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

---

## FilterSobelHorizGetBufferSize

*Computes the size of the external buffer for the horizontal Sobel filter with border.*

---

### Syntax

```
IppStatus ippiFilterSobelHorizGetBufferSize_8u16s_C1R(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);
IppStatus ippiFilterSobelHorizGetBufferSize_32f_C1R(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);
```

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

### Description

The function `ippiFilterSobelHorizGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function [ippiFilterSobelHorizBorder](#). The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table 9-3](#)). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterSobelVertGetBufferSize, FilterSobelNegVertGetBufferSize

*Computes the size of the external buffer for the vertical Sobel filter with border.*

---

### Syntax

```
IppStatus ippiFilterSobelVertGetBufferSize_8u16s_C1R(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);

IppStatus ippiFilterSobelVertGetBufferSize_32f_C1R(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);

IppStatus ippiFilterSobelNegVertGetBufferSize_8u16s_C1R(IppiSize
    roiSize, IppiMaskSize mask, int* pBufferSize);

IppStatus ippiFilterSobelNegVertGetBufferSize_32f_C1R(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);
```

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

### Description

The functions `ippiFilterSobelVertGetBufferSize` and `ippiFilterSobelNegVertGetBufferSize` are declared in the `ippcv.h` file. These functions compute the size of the external buffer that is required for the filter functions [ippiFilterSobelVertBorder](#), [FilterSobelNegVertBorder](#), and [ippiFilterSobelNegVertBorder](#) respectively. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table 9-3](#)). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is NULL.

---

<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

---

## FilterSobelHorizSecondGetBufferSize

*Computes the size of the external buffer for the second derivative horizontal Sobel filter with border.*

---

### Syntax

```
IppStatus ippFilterSobelHorizSecondGetBufferSize_8u16s_C1R(IppiSize
    roiSize, IppiMaskSize mask, int* pBufferSize);
IppStatus ippFilterSobelHorizSecondGetBufferSize_32f_C1R(IppiSize
    roiSize, IppiMaskSize mask, int* pBufferSize);
```

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

### Description

The function `ippFilterSobelHorizSecondGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function [ippFilterSobelHorizSecondBorder](#). The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table 9-3](#)). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

`ippStsMaskSizeErr` Indicates an error condition if *mask* has a wrong value.

---

## FilterSobelVertSecondGetBufferSize

*Computes the size of the external buffer for the second derivative vertical Sobel filter with border.*

---

### Syntax

```
IppStatus ippFilterSobelVertSecondGetBufferSize_8u16s_C1R(IppiSize
    roiSize, IppiMaskSize mask, int* pBufferSize);
IppStatus ippFilterSobelVertSecondGetBufferSize_32f_C1R(IppiSize
    roiSize, IppiMaskSize mask, int* pBufferSize);
```

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

### Description

The function `ippFilterSobelVertSecondGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function [ippFilterSobelVertSecondBorder](#). The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table 9-3](#)). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.



---

## FilterSobelCrossGetBufferSize

*Computes the size of the external buffer for the cross Sobel filter with border.*

---

### Syntax

```
IppStatus ippiFilterSobelCrossGetBufferSize_8u16s_C1R(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);
IppStatus ippiFilterSobelCrossGetBufferSize_32f_C1R(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);
```

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

### Description

The function `ippiFilterSobelCrossGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function [ippiFilterSobelCrossBorder](#). The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table 9-3](#)). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## FilterLaplacianGetBufferSize

*Computes the size of the external buffer for the Laplace filter with border.*

---

### Syntax

```
IppStatus ippiFilterLaplacianGetBufferSize_8u16s_C1R(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);
IppStatus ippiFilterLaplacianGetBufferSize_32f_C1R(IppiSize roiSize,
    IppiMaskSize mask, int* pBufferSize);
```

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

### Description

The function `ippiFilterLaplacianGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function [ippiFilterLaplacianBorder](#). The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table 9-3](#)). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

---

## FilterLowpassGetBufferSize

*Computes the size of the external buffer for the lowpass filter with border.*

---

### Syntax

```
IppStatus ippiFilterLowpassGetBufferSize_8u_C1R(IppiSize roiSize,
        IppiMaskSize mask, int* pBufferSize);
IppStatus ippiFilterLowpassGetBufferSize_32f_C1R(IppiSize roiSize,
        IppiMaskSize mask, int* pBufferSize);
```

### Parameters

<i>roiSize</i>	Maximum size of the source and destination image ROI.
<i>mask</i>	Predefined mask of <code>IppiMaskSize</code> type.
<i>pBufferSize</i>	Pointer to the buffer size.

### Description

The function `ippiFilterLowpassGetBufferSize` is declared in the `ippcv.h` file. This function computes the size of the external buffer that is required for the filter function [ippiFilterLowpassBorder](#). The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask* (see [Table 9-3](#)). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of *roiSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsMaskSizeErr</code>	Indicates an error condition if <i>mask</i> has a wrong value.

## GenSobelKernel

*Computes kernel for the Sobel filter.*

---

### Syntax

```
IppStatus ippiGenSobelKernel_16s(Ipp16s* pDst, int kernelSize, int dx,
    int sign);
IppStatus ippiGenSobelKernel_32f(Ipp132f* pDst, int kernelSize, int dx,
    int sign);
```

### Parameters

<i>pDst</i>	Pointer to the destination vector.
<i>kernelSize</i>	Size of the Sobel kernel.
<i>dx</i>	Order of derivative.
<i>sign</i>	Specifies signs of kernel elements.

### Description

The function `ippiGenSobelKernel` is declared in the `ippcv.h` file. This function computes the one dimensional Sobel kernel. Kernel coefficients are equal to coefficients of the polynomial

$$(1 + x)^{kernelSize - dx - 1} \cdot (x - 1)^{dx}$$

If the *sign* parameter is negative, then signs of kernel coefficients are changed. Kernel calculated by this function can be used to filter images by high order Sobel filter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>kernelSize</i> is less than 3 or is even.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>dx</i> is equal to or less than <i>kernelSize</i> , or <i>dx</i> is negative.

## FilterScharrHorizBorder

*Applies horizontal Scharr filter with border.*

### Syntax

```
IppStatus ippiFilterScharrHorizBorder_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterScharrHorizBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination image ROI.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table data-bbox="565 999 1347 1164"> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippBorderRepl</code>	Replicated border is used.										
<code>ippBorderWrap</code>	Wrapped border is used										
<code>ippBorderMirror</code>	Mirrored border is used										
<code>ippBorderMirrorR</code>	Mirrored border with replication is used										
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).										
<i>pBuffer</i>	Pointer to the working buffer.										

### Description

The function `ippiFilterScharrHorizBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the horizontal Scharr filter (y-derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. For the function flavor `ippiFilterScharrHorizBorder_32f_C1R` the source image

can be used as the destination image.

The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is a matrix of 3x3 size with the anchor in the center cell (red) and the following values:

```

3      10      3
0      0      0
-3     -10     -3

```

The function requires the working buffer *pBuffer* whose size should be computed by the function [ippiFilterScharrHorizGetBufferSize](#) beforehand.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has wrong value.

---

## FilterScharrVertBorder

*Applies vertical Scharr filter with border.*

---

### Syntax

```

IppStatus ippiFilterScharrVertBorder_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterScharrVertBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType
    borderType, Ipp32f borderValue, Ipp8u* pBuffer);

```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination image ROI.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippBorderRepl</code>	Replicated border is used.										
<code>ippBorderWrap</code>	Wrapped border is used										
<code>ippBorderMirror</code>	Mirrored border is used										
<code>ippBorderMirrorR</code>	Mirrored border with replication is used										
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).										
<i>pBuffer</i>	Pointer to the working buffer.										

## Description

The function `ippiFilterScharrVertBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the vertical Scharr filter (x-derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. For the function flavor `ippiFilterScharrVertBorder_32f_C1R` the source image can be used as the destination image. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is a matrix of 3x3 size with the anchor in the center cell (red) and the following values:

3	0	-3
10	0	-10
3	0	-3

The function requires the working buffer *pBuffer* whose size should be computed by the function [ippiFilterScharrVertGetBufferSize](#) beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <pixelsize>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has wrong value.

---

## FilterSobelHorizBorder

*Applies horizontal Sobel filter with border.*

---

### Syntax

```

IppStatus ippFilterSobelHorizBorder_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippFilterSobelHorizBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);

```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.				
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.				
<i>pDst</i>	Pointer to the destination image ROI.				
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.				
<i>roiSize</i>	Size of the source and destination image ROI.				
<i>mask</i>	Type of the filter kernel.				
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.
<code>ippBorderConst</code>	Values of all border pixels are set to constant.				
<code>ippBorderRepl</code>	Replicated border is used.				



	<code>ippBorderWrap</code>	Wrapped border is used
	<code>ippBorderMirror</code>	Mirrored border is used
	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>borderValue</code>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).	
<code>pBuffer</code>	Pointer to the working buffer.	

## Description

The function `ippiFilterSobelHorizBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the horizontal Sobel filter (y-derivative) to the source image `pSrc` and stores results to the destination image of the same size `pDst`. For the function flavor `ippiFilterSobelHorizBorder_32f_C1R` the source image can be used as the destination image. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The kernels have the following values with the anchor in the center cell (red):

					1	4	6	4	1
1	2	1			2	8	12	8	2
0	0	0	or		0	0	0	0	0
-1	-2	-1			-2	-8	-12	-8	-4
					-1	-4	-6	-4	-1

The function requires the working buffer `pBuffer` whose size should be computed by the function [`ippiFilterSobelHorizGetBufferSize`](#) beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBorderErr</code>	Indicates an error condition if <code>borderType</code> has wrong value.

`ippStsMaskErr` Indicates an error condition if *mask* has wrong value.

---

## FilterSobelVertBorder, FilterSobelNegVertBorder,

*Applies vertical Sobel filter with border.*

---

### Syntax

```
IppStatus ippiFilterSobelVertBorder_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterSobelVertBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterSobelNegVertBorder_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterSobelNegVertBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination image ROI.
<i>mask</i>	Type of the filter kernel.
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <div> <div><code>ippBorderConst</code></div> <div>Values of all border pixels are set to constant.</div> </div> <div> <div><code>ippBorderRepl</code></div> <div>Replicated border is used.</div> </div>

	<code>ippBorderWrap</code>	Wrapped border is used
	<code>ippBorderMirror</code>	Mirrored border is used
	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>borderValue</code>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).	
<code>pBuffer</code>	Pointer to the working buffer.	

## Description

The function `ippiFilterSobelVertBorder` and `ippiFilterSobelNegVertBorder` are declared in the `ippcv.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)). These functions apply the vertical Sobel filter (x-derivative) to the source image ROI `pSrc` and stores results to the destination image ROI of the same size `pDst`. Source image can be used as the destination image if they have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The anchor cell is the center cell of the kernel (red).

The function `ippiFilterSobelVertBorder` uses the kernels with the following the following coefficients:

				-1	-2	0	2	1
-1	0	1		-4	-8	0	8	4
-2	0	2	or	-6	-12	0	12	6
-1	0	1		-4	-8	0	8	4
				-1	-2	0	2	1

The function `ippiFilterSobelNegVertBorder` uses the kernels which coefficients are the same in magnitude but opposite in sign:

				1	2	0	-2	-1
1	0	-1		4	8	0	-8	-4
2	0	-2	or	6	12	0	-12	-6
1	0	-1		4	8	0	-8	-4
				1	2	0	-2	-1

Both functions require the working buffer `pBuffer` whose size must be computed beforehand by the functions [`ippiFilterSobelVertGetBufferSize`](#), [`FilterSobelNegVertGetBufferSize`](#) and [`ippiFilterSobelNegVertGetBufferSize`](#) respectively.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <pixelsize>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has wrong value.
<code>ippStsMaskErr</code>	Indicates an error condition if <i>mask</i> has wrong value.

---

## FilterSobelHorizSecondBorder

*Applies horizontal (second derivative) Sobel filter with border.*

---

### Syntax

```

IppStatus ippiFilterSobelHorizSecondBorder_8u16s_C1R(const Ipp8u* pSrc,
    int srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize,
    IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue,
    Ipp8u* pBuffer);

IppStatus ippiFilterSobelHorizSecondBorder_32f_C1R(const Ipp32f* pSrc,
    int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize,
    IppiMaskSize mask, IppiBorderType borderType, Ipp32f borderValue,
    Ipp8u* pBuffer);

```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

<i>roiSize</i>	Size of the source and destination image ROI.
<i>mask</i>	Type of the filter kernel.
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: ippBorderConst      Values of all border pixels are set to constant. ippBorderRepl      Replicated border is used. ippBorderWrap      Wrapped border is used ippBorderMirror     Mirrored border is used ippBorderMirrorR    Mirrored border with replication is used
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
<i>pBuffer</i>	Pointer to the working buffer.

## Description

The function `ippiFilterSobelHorizSecondBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the second derivative horizontal Sobel filter (y-derivative) to the source image *pSrc* and stores results to the destination image of the same size *pDst*. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the *borderType* and *borderValue* parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter *mask*. The kernels have the following values with the anchor in the center cell (red):

$$\begin{array}{ccccc}
 & & & 1 & 4 & 6 & 4 & 1 \\
 & & & 0 & 0 & 0 & 0 & 0 \\
 & 1 & 2 & 1 & & & & \\
 -2 & -4 & -2 & \text{or} & -2 & -8 & -12 & -8 & -2 \\
 & 1 & 2 & 1 & & & & & \\
 & & & & 1 & 4 & 6 & 4 & 1
 \end{array}$$

The function requires the working buffer *pBuffer* whose size should be computed by the function [ippiFilterSobelHorizSecondGetBufferSize](#) beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBorderErr</code>	Indicates an error condition if <i>borderType</i> has wrong value.
<code>ippStsMaskErr</code>	Indicates an error condition if <i>mask</i> has wrong value.

---

## FilterSobelVertSecondBorder

*Applies vertical (second derivative) Sobel filter with border.*

---

### Syntax

```
IppStatus ippiFilterSobelVertSecondBorder_8u16s_C1R(const Ipp8u* pSrc,
    int srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize,
    IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue,
    Ipp8u* pBuffer);

IppStatus ippiFilterSobelVertSecondBorder_32f_C1R(const Ipp32f* pSrc,
    int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize,
    IppiMaskSize mask, IppiBorderType borderType, Ipp32f borderValue,
    Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.				
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.				
<i>pDst</i>	Pointer to the destination image ROI.				
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.				
<i>roiSize</i>	Size of the source and destination image ROI.				
<i>mask</i>	Type of the filter kernel.				
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.
<code>ippBorderConst</code>	Values of all border pixels are set to constant.				
<code>ippBorderRepl</code>	Replicated border is used.				

	<code>ippBorderWrap</code>	Wrapped border is used
	<code>ippBorderMirror</code>	Mirrored border is used
	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>borderValue</code>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).	
<code>pBuffer</code>	Pointer to the working buffer.	

## Description

The function `ippiFilterSobelVertSecondBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the second derivative vertical Sobel filter (x-derivative) to the source image `pSrc` and stores results to the destination image of the same size `pDst`. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The kernels have the following values with the anchor in the center cell (red):

				1	0	-2	0	1
1	-2	1		4	0	-8	0	4
2	-4	2	or	6	0	-12	0	6
1	-2	1		4	0	-8	0	4
				1	0	-2	0	1

The function requires the working buffer `pBuffer` whose size should be computed by the function [`ippiFilterSobelVertSecondGetBufferSize`](#) beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBorderErr</code>	Indicates an error condition if <code>borderType</code> has wrong value.

`ippStsMaskErr` Indicates an error condition if *mask* has wrong value.

---

## FilterSobelCrossBorder

*Applies second derivative cross Sobel filter with border.*

---

### Syntax

```
ippStatus ippiFilterSobelCrossBorder_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

ippStatus ippiFilterSobelCrossBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination image ROI.										
<i>mask</i>	Type of the filter kernel.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table data-bbox="633 1140 1404 1306"> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippBorderRepl</code>	Replicated border is used.										
<code>ippBorderWrap</code>	Wrapped border is used										
<code>ippBorderMirror</code>	Mirrored border is used										
<code>ippBorderMirrorR</code>	Mirrored border with replication is used										
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).										
<i>pBuffer</i>	Pointer to the working buffer.										



## Description

The function `ippiFilterSobelCrossBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the second derivative cross Sobel filter (xy-derivative) to the source image `pSrc` and stores results to the destination image of the same size `pDst`. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The kernels have the following values with the anchor in the center cell (red):

-1	0	1		-1	-2	0	2	1
0	0	0	or	-2	-4	0	4	2
1	0	-1		0	0	0	0	0
				2	4	0	-4	-2
				1	2	0	-2	-1

The function requires the working buffer `pBuffer` whose size should be computed by the function [ippiFilterSobelCrossGetBufferSize](#) beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBorderErr</code>	Indicates an error condition if <code>borderType</code> has wrong value.
<code>ippStsMaskErr</code>	Indicates an error condition if <code>mask</code> has wrong value.

## FilterLaplacianBorder

*Applies Laplacian filter with border.*

---

### Syntax

```

IppStatus ippiFilterLaplacianBorder_8u16s_C1R(const Ipp8u* pSrc, int
    srcStep, Ipp16s* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterLaplacianBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);

```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination image ROI.										
<i>mask</i>	Type of the filter kernel.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippBorderRepl</code>	Replicated border is used.										
<code>ippBorderWrap</code>	Wrapped border is used										
<code>ippBorderMirror</code>	Mirrored border is used										
<code>ippBorderMirrorR</code>	Mirrored border with replication is used										
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).										
<i>pBuffer</i>	Pointer to the working buffer.										

## Description

The function `ippiFilterLaplacianBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the laplacian filter to the source image `pSrc` and stores results to the destination image of the same size `pDst`. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The kernels have the following values with the anchor in the center cell (red):

2	0	2		2	4	4	4	2
0	-8	0	or	4	0	-8	0	4
2	0	2		4	-8	-24	-8	4
				4	0	-8	0	4
				2	4	4	4	2

The function requires the working buffer `pBuffer` whose size should be computed by the function [ippiFilterLaplacianGetBufferSize](#) beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBorderErr</code>	Indicates an error condition if <code>borderType</code> has wrong value.
<code>ippStsMaskErr</code>	Indicates an error condition if <code>mask</code> has wrong value.

## FilterLowpassBorder

*Applies lowpass filter with border.*

---

### Syntax

```

IppStatus ippiFilterLowpassBorder_8u_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
    IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterLowpassBorder_32f_C1R(const Ipp32f* pSrc, int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize
    mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);

```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.										
<i>pDst</i>	Pointer to the destination image ROI.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.										
<i>roiSize</i>	Size of the source and destination image ROI.										
<i>mask</i>	Type of the filter kernel.										
<i>borderType</i>	Type of border (see <a href="#">Borders</a> ); following values are possible: <table data-bbox="633 1050 1404 1212"> <tr> <td><code>ippBorderConst</code></td><td>Values of all border pixels are set to constant.</td></tr> <tr> <td><code>ippBorderRepl</code></td><td>Replicated border is used.</td></tr> <tr> <td><code>ippBorderWrap</code></td><td>Wrapped border is used</td></tr> <tr> <td><code>ippBorderMirror</code></td><td>Mirrored border is used</td></tr> <tr> <td><code>ippBorderMirrorR</code></td><td>Mirrored border with replication is used</td></tr> </table>	<code>ippBorderConst</code>	Values of all border pixels are set to constant.	<code>ippBorderRepl</code>	Replicated border is used.	<code>ippBorderWrap</code>	Wrapped border is used	<code>ippBorderMirror</code>	Mirrored border is used	<code>ippBorderMirrorR</code>	Mirrored border with replication is used
<code>ippBorderConst</code>	Values of all border pixels are set to constant.										
<code>ippBorderRepl</code>	Replicated border is used.										
<code>ippBorderWrap</code>	Wrapped border is used										
<code>ippBorderMirror</code>	Mirrored border is used										
<code>ippBorderMirrorR</code>	Mirrored border with replication is used										
<i>borderValue</i>	The constant value to assign to the pixels in the constant border (not applicable for other border's type).										
<i>pBuffer</i>	Pointer to the working buffer.										

## Description

The function `ippiFilterLowpassBorder` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function applies the lowpass filter (blur operation) to the source image `pSrc` and stores results to the destination image of the same size `pDst`. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The anchor cell is the center cell (red) of the kernel.

The 3x3 filter uses the kernel with the following values:

```
1/9  1/9  1/9
1/9  1/9  1/9
1/9  1/9  1/9
```

The 5x5 filter uses the kernel with the following values:

```
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
```

The function requires the working buffer `pBuffer` whose size should be computed by the function [ippiFilterLowpassGetBufferSize](#) beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBorderErr</code>	Indicates an error condition if <code>borderType</code> has wrong value.
<code>ippStsMaskErr</code>	Indicates an error condition if <code>mask</code> has wrong value.



# Image Linear Transforms

This chapter describes the Intel IPP functions that perform linear transform operations on an image buffer.

These operations include Fast Fourier Transform (FFT), Discrete Fourier Transform (DFT), and Discrete Cosine Transform (DCT).

[Table 10-1](#) lists the Intel IPP linear transform functions.:

**Table 10-1      Image Linear Transform Functions**

Function Base Name	Operation
<b>Fourier Transforms</b>	
<a href="#">FFTInitAlloc</a>	Allocates memory and fills in context data needed for the image FFT functions to operate.
<a href="#">FFTFree</a>	Deallocates memory used by the FFT context structure.
<a href="#">FFTGetBufSize</a>	Determines the size of an external work buffer that can be used by the FFT functions.
<a href="#">FFTFwd</a>	Applies forward Fast Fourier Transform to an image.
<a href="#">FFTInv</a>	Applies inverse Fast Fourier Transform to complex source data and stores results in a destination image.
<a href="#">DFTInitAlloc</a>	Allocates memory and fills in context data needed for the image DFT functions to operate.
<a href="#">DFTFree</a>	Deallocates memory used by the DFT context structure.
<a href="#">DFTGetBufSize</a>	Determines the size of an external work buffer that can be used by the DFT functions.
<a href="#">DFTFwd</a>	Applies forward Discrete Fourier Transform to an image.

continued

**Table 10-1 Image Linear Transform Functions (continued)**

Function Base Name	Operation
<a href="#"><u>DFTInv</u></a>	Applies inverse Discrete Fourier Transform to complex sourcedataandstoresresultsinadestinationimage
<a href="#"><u>MulPack</u></a>	Multiplies two source images with data in packed format and stores the result a destination image in packed format.
<a href="#"><u>MulPackConj</u></a>	Multiplies two source images with data in packed format and stores the result in a destination image that is complex-conjugate to that obtained with the <code>ippiMulPack</code> function.
<a href="#"><u>Magnitude</u></a>	Computes the magnitude of elements of complex data images.
<a href="#"><u>MagnitudePack</u></a>	Computes the magnitude of elements of an image in packed format.
<a href="#"><u>Phase</u></a>	Computes the phase of elements of complex data images.
<a href="#"><u>PhasePack</u></a>	Computes the phase of elements of an image in packed format.
<a href="#"><u>PolarToCart</u></a>	Converts an image in the polar coordinate form to Cartesian coordinate form.
<a href="#"><u>PackToCplxExtend</u></a>	Converts an image in packed format to a complex data image.
<a href="#"><u>WinBartlett</u></a> , <a href="#"><u>WinBartlettSep</u></a>	Applies Bartlett window function to the image.
<a href="#"><u>WinHamming</u></a> , <a href="#"><u>WinHammingSep</u></a>	Applies Hamming window function to the image.
<b>Discrete Cosine Transforms</b>	
<a href="#"><u>DCTFwdInitAlloc</u></a>	Allocates memory and fills in context data needed for the forward DCT function to operate
<a href="#"><u>DCTInvInitAlloc</u></a>	Allocates memory and fills in context data needed for the inverse DCT function to operate
<a href="#"><u>DCTFwdFree</u></a>	Deallocates memory used by the forward DCT context structure.
<a href="#"><u>DCTInvFree</u></a>	Deallocates memory used by the inverse DCT context structure.
<a href="#"><u>DCTFwdGetBufSize</u></a>	Determines the size of an external work buffer that can be used by the forward DCT function
<a href="#"><u>DCTInvGetBufSize</u></a>	Determines the size of an external work buffer that can be used by the inverse DCT function



**Table 10-1** Image Linear Transform Functions (continued)

Function Base Name	Operation
<a href="#">DCTFwd</a>	Performs a forward DCT of an image
<a href="#">DCTInv</a>	Performs an inverse DCT of an image
<a href="#">DCT8x8Fwd</a>	Performs a forward DCT on a buffer of 8x8 size
<a href="#">DCT8x8Inv</a>	Performs an inverse DCT on a buffer of 8x8 size
<a href="#">DCT8x8FwdLS</a>	Performs a forward DCT on a 2D buffer of 8x8 size with level shift
<a href="#">DCT8x8InvLSClip</a>	Performs an inverse DCT on a buffer of 8x8 size with level shift
<a href="#">DCT8x8Inv_2x2</a>	Perform an inverse DCT on a top left quadrant 2x2 or 4x4 of the buffer of 8x8 size
<a href="#">DCT8x8Inv_4x4</a>	

To speed up performance, linear transform functions use precomputed auxiliary data that is needed for computation of the transforms (that is, tables of twiddle factors for FFT functions). This data is calculated by the respective initialization functions and passed to the transform functions in context structures specific for each type of transform.

Most linear transform functions in Intel IPP have code branches that implement different algorithms to compute the results. You can choose the desired code variety to be used by the transform function by setting the *hint* argument to one of the following values that are listed in [Table 10-2](#) :

**Table 10-2** Hint Arguments for Linear Transform Functions

Value	Description
<code>ippAlgHintNone</code>	The computation algorithm will be chosen by the internal function logic.
<code>ippAlgHintFast</code>	Fast algorithm must be used. The output results will be less accurate.
<code>ippAlgHintAccurate</code>	High accuracy algorithm must be used. The function will need more time to execute.

Intel IPP linear transform functions can use external work buffers for storing data and intermediate results, which eliminates the need to allocate and free internal memory buffers and thus helps to further increase function performance. To determine the required work buffer size, use one of the respective support functions specific for each transform type. In case when no external buffer is specified, the transform functions handle memory allocation internally.

All Intel IPP linear transform functions except DCT of 8x8 size work on images with floating point data only.

## Fourier Transforms

Intel IPP functions that compute FFT and DFT can process both real and complex images. Function flavors operating on real data are distinguished by R suffix present in function-specific modifier of their full name, whereas complex flavors' names include C suffix (see [Function Naming](#) in Chapter 2).

The results of computing the Fourier transform can be normalized by specifying the appropriate value of *flag* argument for context initialization. This parameter sets up a pair of matched normalization factors to be used in forward and inverse transforms as listed in the following table:

**Table 10-3 Normalization Factors for Fourier Transform Results**

Value of <i>flag</i> Argument	Normalization Factors	
	Forward Transform	Inverse Transform
IPP_FFT_DIV_FWD_BY_N	1/MN	1
IPP_FFT_DIV_INV_BY_N	1	1/MN
IPP_FFT_DIV_BY_SQRTN	1/sqrt (MN)	1/sqrt (MN)
IPP_FFT_NODIV_BY_ANY	1	1

In this table, N and M denote the length of Fourier transform in the x- and y-directions, respectively (or, equivalently, the number of columns and rows in the 2D array being transformed). For the FFT, these lengths must be integer powers of 2, that is  $N=2^{\text{orderX}}$ ,  $M=2^{\text{orderY}}$ , where power exponents are known as order of FFT. For the DFT, N and M can take on arbitrary integer non-negative values.

## Real - Complex Packed (RCPack2D) Format

The forward Fourier transform of a real two-dimensional image data yields a matrix of complex results which has conjugate-symmetric properties. Intel IPP functions use packed format RCPack2D for storing and retrieving data of this type. Accordingly, real flavors of the inverse Fourier transform functions convert packed complex conjugate-symmetric data back to its real origin.

The RCPack2D format exploits the complex conjugate symmetry of the transformed data to store only a half of the resulting Fourier coefficients. For the  $N$  by  $M$  transform, the respective FFT and DFT functions actually store real and imaginary parts of the complex Fourier coefficients  $A(i, j)$  for  $i = 0, \dots, M-1$ ;  $j = 0, \dots, N/2$  in a single real array of dimensions  $(N, M)$ . The RCPack2D storage format is slightly different for odd and even  $M$  and is arranged in accordance with the following tables:

**Table 10-4 RCPack2D Storage for Odd Number of Rows**

Re $A(0,0)$	Re $A(0,1)$	Im $A(0,1)$	...	Re $A(0,(N-1)/2)$	Im $A(0,(N-1)/2)$	Re $A(0,N/2)$
Re $A(1,0)$	Re $A(1,1)$	Im $A(1,1)$	...	Re $A(1,(N-1)/2)$	Im $A(1,(N-1)/2)$	Re $A(1,N/2)$
Im $A(1,0)$	Re $A(2,1)$	Im $A(2,1)$	...	Re $A(2,(N-1)/2)$	Im $A(2,(N-1)/2)$	Im $A(1,N/2)$
...	...	...	...	...	...	...
Re $A(M/2,0)$	Re $A(M-2,1)$	Im $A(M-2,1)$	...	Re $A(M-2,(N-1)/2)$	Im $A(M-2,(N-1)/2)$	Re $A(M/2,N/2)$
Im $A(M/2,0)$	Re $A(M-1,1)$	Im $A(M-1,1)$	...	Re $A(M-1,(N-1)/2)$	Im $A(M-1,(N-1)/2)$	Im $A(M/2,N/2)$

**Table 10-5 RCPack2D Storage for Even Number of Rows**

Re $A(0,0)$	Re $A(0,1)$	Im $A(0,1)$	...	Re $A(0,(N-1)/2)$	Im $A(0,(N-1)/2)$	Re $A(0,N/2)$
Re $A(1,0)$	Re $A(1,1)$	Im $A(1,1)$	...	Re $A(1,(N-1)/2)$	Im $A(1,(N-1)/2)$	Re $A(1,N/2)$
Im $A(1,0)$	Re $A(2,1)$	Im $A(2,1)$	...	Re $A(2,(N-1)/2)$	Im $A(2,(N-1)/2)$	Im $A(1,N/2)$
...	...	...	...	...	...	...
Re $A(M/2-1,0)$	Re $A(M-3,1)$	Im $A(M-3,1)$	...	Re $A(M-3,(N-1)/2)$	Im $A(M-3,(N-1)/2)$	Re $A(M/2-1,N/2)$
Im $A(M/2-1,0)$	Re $A(M-2,1)$	Im $A(M-2,1)$	...	Re $A(M-2,(N-1)/2)$	Im $A(M-2,(N-1)/2)$	Im $A(M/2-1,N/2)$
Re $A(M/2,0)$	Re $A(M-1,1)$	Im $A(M-1,1)$	...	Re $A(M-1,(N-1)/2)$	Im $A(M-1,(N-1)/2)$	Re $A(M/2,N/2)$

The shaded columns to the right side of the tables indicate values for even  $N$  only.

Note the above tables show the arrangement of coefficients for one channel. For multichannel images the channel coefficients are clustered and stored consecutively, for example, for 3-channel image they are stored in the following way: C1-Re A(0,0); C2-Re A(0,0); C3-Re A(0,0); C1-Re A(0,1); C2-Re A(0,1); C3-Re A(0,1); C1-Im A(0,1); C2-Im A(0,1); ...

The remaining Fourier coefficients are obtained using the following relationships based on conjugate-symmetric properties:

$$\begin{aligned} A(i, j) &= \text{conj}(A(M-i, N-j)) & i &= 1, \dots, M-1; & j &= 1, \dots, N-1 \\ A(0, j) &= \text{conj}(A(0, N-j)) & j &= 1, \dots, N-1 \\ A(i, 0) &= \text{conj}(A(M-i, 0)) & i &= 1, \dots, M-1 \end{aligned}$$

---

## FFTInitAlloc

*Allocates memory and fills in context data needed for the image FFT functions to operate.*

---

### Syntax

```
IppStatus ippiFFTInitAlloc_R_32s (IppiFFTSpec_R_32s** pFFTSpec,
    int orderX, int orderY, int flag, IppHintAlgorithm hint);
IppStatus ippiFFTInitAlloc_R_32f (IppiFFTSpec_R_32f** pFFTSpec,
    int orderX, int orderY, int flag, IppHintAlgorithm hint);
IppStatus ippiFFTInitAlloc_C_32fc (IppiFFTSpec_C_32fc** pFFTSpec,
    int orderX, int orderY, int flag, IppHintAlgorithm hint);
```

### Parameters

<i>pFFTSpec</i>	Pointer to the pointer to the FFT context structure being initialized.
<i>orderX, orderY</i>	Order of the FFT in x- and y- directions, respectively.
<i>flag</i>	Flag to choose the results normalization option.
<i>hint</i>	Option to select the algorithmic implementation of the transform function (see <a href="#">Table 10-2</a> ).

### Description

The function `ippiFFTInitAlloc` is declared in the `ippi.h` file. This function allocates memory and initializes the context structure *pFFTSpec* needed to compute the forward and inverse FFT of a two-dimensional image data.

The [ippiFFTWd](#) and [ippiFFTInv](#) functions called with the pointer to the initialized *pFFTSpec* structure as an argument will compute the fast Fourier transform with the following characteristics:

- length  $N=2^{orderX}$  in x-direction by  $M=2^{orderY}$  in y-direction
- results normalization mode as set by *flag* (see [Table 10-3](#))
- computation algorithm indicated by *hint*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pFFTSpec</i> pointer is NULL.
<code>ippStsFftOrderErr</code>	Indicates an error condition if the FFT order value is illegal.
<code>ippStsFFTFlagErr</code>	Indicates an error condition if <i>flag</i> has an illegal value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## FFTFree

*Deallocates memory used by the FFT context structure.*

---

### Syntax

```
IppStatus ippiFFTFree_R_32s (IppiFFTSpec_R_32s* pFFTSpec);
IppStatus ippiFFTFree_R_32f (IppiFFTSpec_R_32f* pFFTSpec);
IppStatus ippiFFTFree_C_32fc (IppiFFTSpec_C_32fc* pFFTSpec);
```

### Parameters

*pFFTSpec*                      Pointer to the FFT context structure that has to be released.

### Description

The function `ippiFFTFree` is declared in the `ippi.h` file. This function releases memory used by the previously initialized FFT context structure *pFFTSpec*. Call this function after your application program has finished computing the fast Fourier transforms.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pFFTSpec</code> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <code>pFFTSpec</code> structure is passed.

---

## FFTGetBufSize

*Determines the size of external work buffer that can be used by the FFT functions.*

---

### Syntax

```
IppStatus ippifftGetBufSize_R_32s (IppiFFTSpec_R_32s* pFFTSpec,
                                   int* pSize);
IppStatus ippifftGetBufSize_R_32f (IppiFFTSpec_R_32f* pFFTSpec,
                                   int* pSize);
IppStatus ippifftGetBufSize_C_32fc (IppiFFTSpec_C_32fc* pFFTSpec,
                                   int* pSize);
```

### Parameters

<code>pFFTSpec</code>	Pointer to the previously initialized FFT context structure.
<code>pSize</code>	Pointer to the size of the external work buffer to be used by the FFT functions.

### Description

The function `ippifftGetBufSize` is declared in the `ippi.h` file. This function determines the size in bytes of an external memory buffer that can be used by FFT functions with context `pFFTSpec` as a workspace during calculations.

If you want to use external buffering, call this function after `pFFTSpec` initialization.

Use the computed `pSize` value to set the size in bytes of an external buffer to be allocated by means of, for example, [ippiMalloc](#) functions.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pFFTSpec</code> or <code>pSize</code> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <code>pFFTSpec</code> structure is passed.

---

## FFTFwd

*Applies forward Fast Fourier Transform to an image.*

---

### Syntax

#### Case 1: Not-in-place operation on integer data

```
IppStatus ippifftFwd_RToPack_<mod> (const Ipp8u* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, const IppiFFTSpec_R_32s* pFFTSpec,
    int scaleFactor, Ipp8u* pBuffer);
```

Supported values for `mod`:

```
8u32s_C1RSfs
8u32s_C3RSfs
8u32s_C4RSfs
8u32s_AC4RSfs
```

#### Case 2: Not-in-place operation on floating-point data

```
IppStatus ippifftFwd_RToPack_<mod> (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,
    Ipp8u* pBuffer);
```

Supported values for `mod`:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

**Case 3: Not-in-place operation on complex data**

```
IppStatus ippiFFTFwd_CToC_32fc_C1R (const Ipp32fc* pSrc, int srcStep,  
    Ipp32fc* pDst, int dstStep, const IppiFFTSpec_C_32fc* pFFTSpec,  
    Ipp8u* pBuffer);
```

**Case 4: In-place operation on floating-point data**

```
IppStatus ippiFFTFwd_RToPack_<mod>(Ipp32f* pSrcDst, int srcDstStep,  
    const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1IR  
32f_C3IR  
32f_C4IR  
32f_AC4IR
```

**Case 5: In-place operation on complex data**

```
IppStatus ippiFFTFwd_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep,  
    const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>scaleFactor</i>	The factor for integer result scaling (see <a href="#">“Integer Result Scaling”</a> ).
<i>pBuffer</i>	Pointer to the external buffer to be used by the FFT functions.

**Description**

The function `ippiFFTFwd` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).



This function performs a forward FFT on each channel of the source image ROI *pSrc* (*pSrcDst* for in-place flavors) and writes the Fourier coefficients into the corresponding channel of the destination buffer *pDst* (*pSrcDst* for in-place flavors). The size of ROI is  $N \times M$ , it is specified by the parameters *orderX*, *orderY* calling the function [ippiFFTInitAlloc](#).

The function flavor `ippiFFTFwd_RTToPack` that operates on images with real data takes advantage of the symmetry property and stores the output data in [RCPack2D format](#). It supports processing of the 1-, 3-, and 4-channel images. Note that the functions with `AC4` descriptor do not process alpha channel.

The function flavor `ippiFFTFwd_CTtoC` that operates on images with complex data does not perform any packing of the transform results as no symmetry with respect to frequency domain data is observed in this case. Memory layout of images with complex data follows the same conventions as for real images provided that each pixel value consists of two numbers: imaginary and real part.

The forward FFT functions use the previously initialized *pFFTSpec* context structure to set the mode of calculations and retrieve support data.

The application can also pass the pointer to the external work buffer *pBuffer*. In this case the required buffer size must be determined by calling the [ippiFFTGetBufSize](#) function prior to using the FFT computation functions. If a null pointer is passed, memory allocation is handled by the `ippiFFTFwd` function internally.

The [Example 10-1](#) illustrates the use of the forward FFT function for processing real data.

## Example 10-1 Fast Fourier Transform of a Real Image

---

```
IppStatus fft( void ) {
    Ipp32f src[64] = {0}, dst[64];
    IppiFFTSpec_R_32f *spec;
    IppStatus status;
    src[0] = -3; src[9] = 1;
    ippiFFTInitAlloc_R_32f( &spec, 3, 3, IPP_FFT_DIV_INV_BY_N,
                           ippiAlgHintAccurate );
    status = ippiFFTFwd_RToPack_32f_C1R( src, 8*sizeof(Ipp32f), dst,
                                           8*sizeof(Ipp32f), spec, 0 );

    ippiFFTFree_R_32f( spec );
    return status;
}
```

On exit from the `ippiFFTFwd_RToPack` function, the destination buffer contains the following data in RCPack2D format:

-2.00	-2.29	-0.71	-3.00	-1.00	-3.71	-0.71	-4.00
-2.29	-3.00	-1.00	-3.71	-0.71	-4.00	+0.00	-3.71
-0.71	-3.71	-0.71	-4.00	+0.00	-3.71	+0.71	+0.71
-3.00	-4.00	+0.00	-3.71	+0.71	-3.00	+1.00	-3.00
-1.00	-3.71	+0.71	-3.00	+1.00	-2.29	+0.71	+1.00
-3.71	-3.00	+1.00	-2.29	+0.71	-2.00	+0.00	-2.29
-0.71	-2.29	+0.71	-2.00	+0.00	-2.29	-0.71	+0.71
-4.00	-2.00	+0.00	-2.29	-0.71	-3.00	-1.00	-2.00

---

The [Example 10-2](#) explains FFT processing of complex data. Note that input values are the same as in the previous example, but specified in complex domain.

**Example 10-2 Fast Fourier Transform of a Complex Image**

```

IppStatus fft_cplx( void ) {
    Ipp32fc src[64] = {0}, dst[64], m3 = {-3,0}, one = {1,0};
    IppiFFTSpec_C_32fc *spec;
    IppStatus status;
    src[0] = m3; src[9] = one;
    ippiFFTInitAlloc_C_32fc( &spec, 3, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintAccurate);
    status = ippiFFTForward_CToC_32fc_C1R( src, 8*sizeof(Ipp32fc), dst, 8*sizeof(Ipp32fc),
                                           spec, 0 );

    ippiFFTFree_C_32fc( spec );
    return status;
}

```

On exit from the `ippiFFTForward_CToC` function, the destination buffer contains the following data (real and imaginary part of complex numbers are given in comma-delimited format):

```

-2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7
-2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0,-0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0, 0.0
-3.0,-1.0 -3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0,-0.0 -2.3,-0.7
-3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0,-0.0 -2.3,-0.7 -3.0,-1.0
-4.0, 0.0 -3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7
-3.7, 0.7 -3.0, 1.0 -2.3, 0.7 -2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0,-0.0
-3.0, 1.0 -2.3, 0.7 -2.0, 0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0,-0.0 -3.7, 0.7
-2.3, 0.7 -2.0,-0.0 -2.3,-0.7 -3.0,-1.0 -3.7,-0.7 -4.0, 0.0 -3.7, 0.7 -3.0, 1.0

```

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pFFTSpec</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pFFTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## FFTInv

*Applies an inverse FFT to complex source data and stores results in a destination image.*

---

### Syntax

#### Case 1: Not-in-place operation on integer data

```
IppStatus ippiFFTInv_PackToR_<mod>(const Ipp32s* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, const IppiFFTSpec_R_32s* pFFTSpec,  
    int scaleFactor, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32s8u_C1RSfs  
32s8u_C3RSfs  
32s8u_C4RSfs  
32s8u_AC4RSfs
```

#### Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiFFTInv_PackToR_<mod>(const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec,  
    Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1R  
32f_C3R  
32f_C4R  
32f_AC4R
```

#### Case 3: Not-in-place operation on complex data

```
IppStatus ippiFFTInv_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep,  
    Ipp32fc* pDst, int dstStep, const IppiFFTSpec_C_32fc* pFFTSpec,  
    Ipp8u* pBuffer);
```

#### Case 4: In-place operation on floating-point data

```
IppStatus ippiFFTInv_PackToR_<mod>(Ipp32f* pSrcDst, int srcDstStep,  
    const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1IR
```

```

32f_C3IR
32f_C4IR
32f_AC4IR

```

### Case 5: In-place operation on complex data

```

IppStatus ippiFFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep,
    const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);

```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>pFFTSpec</i>	Pointer to the previously initialized FFT context structure.
<i>scaleFactor</i>	The factor for integer result scaling (see <a href="#">“Integer Result Scaling”</a> ).
<i>pBuffer</i>	Pointer to the external buffer to be used by the FFT functions.

### Description

The function `ippiFFTInv` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs an inverse FFT on each channel of the source image *pSrc* (*pSrcDst* for in-place flavors) and writes the restored image data into the corresponding channel of the destination image buffer *pDst* (*pSrcDst* for in-place flavors). The size of ROI is  $N \times M$ , it is specified by the parameters *orderX*, *orderY* calling the function [ippiFFTInitAlloc](#). For function flavor `ippiFFTInv_PackToR`, the input buffer must contain data in [RCPack2D format](#).

The inverse FFT functions use the previously initialized *pFFTSpec* context structure to set the mode of calculations and retrieve support data. The application can also pass the pointer to the external work buffer *pBuffer*. In this case the

required buffer size must be determined by calling the [ippiFFTGetBufSize](#) function prior to using the FFT computation functions. If a null pointer is passed, memory allocation will be handled by the `ippiFFTInv` function internally.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pFFTSpec</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pFFTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## DFTInitAlloc

*Allocates memory and fills in context data needed for the image DFT functions to operate.*

---

### Syntax

```
IppStatus ippiDFTInitAlloc_R_32s (IppiDFTSpec_R_32s** pDFTSpec, IppiSize
    roiSize, int flag, IppHintAlgorithm hint);
IppStatus ippiDFTInitAlloc_R_32f (IppiDFTSpec_R_32f** pDFTSpec, IppiSize
    roiSize, int flag, IppHintAlgorithm hint);
IppStatus ippiDFTInitAlloc_C_32fc (IppiDFTSpec_C_32fc** pDFTSpec,
    IppiSize roiSize, int flag, IppHintAlgorithm hint);
```

### Parameters

<i>pDFTSpec</i>	Pointer to pointer to the DFT context structure being initialized.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>flag</i>	Flag to choose the results normalization option.

*hint* Option to select the algorithmic implementation of the transform function (see [Table 10-2](#) ).

## Description

The function `ippiDFTInitAlloc` is declared in the `ippi.h` file. This function allocates memory and initializes the context structure `pDFTSpec` needed to compute the forward and inverse DFT of a two-dimensional image data. The `ippiDFTFwd` and `ippiDFTInv` functions called with the pointer to the initialized `pDFTSpec` structure as an argument will compute the discrete Fourier transform for points in the ROI of size `roiSize`, with results normalization mode set according to `flag` (see [Table 10-3](#)), and computation algorithm indicated by `hint`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pDFTSpec</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsFFTFldErr</code>	Indicates an error condition if <code>flag</code> has an illegal value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## DFTFree

*Deallocates memory used by the DFT context structure.*

---

## Syntax

```
IppStatus ippiDFTFree_R_32s (IppiDFTSpec_R_32s* pDFTSpec);
IppStatus ippiDFTFree_R_32f (IppiDFTSpec_R_32f* pDFTSpec);
IppStatus ippiDFTFree_C_32fc (IppiDFTSpec_C_32fc* pDFTSpec);
```

## Parameters

*pDFTSpec* Pointer to the DFT context structure that has to be released

### Description

The function `ippiDFTFree` is declared in the `ippi.h` file. This function releases memory used by the previously initialized DFT context structure `pDFTSpec`. Call this function after your application program has finished computing the discrete Fourier transforms.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pDFTSpec</code> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <code>pDFTSpec</code> structure is passed.

---

## DFTGetBufSize

*Determines the size of work buffer that can be used by the DFT functions to operate.*

---

### Syntax

```
IppStatus ippiDFTGetBufSize_R_32s(IppiDFTSpec_R_32s* pDFTSpec,  
    int* pSize);  
  
IppStatus ippiDFTGetBufSize_R_32f(IppiDFTSpec_R_32f* pDFTSpec,  
    int* pSize);  
  
IppStatus ippiDFTGetBufSize_C_32fc(IppiDFTSpec_C_32fc* pDFTSpec,  
    int* pSize);
```

### Parameters

<code>pDFTSpec</code>	Pointer to the previously initialized DFT context structure.
<code>pSize</code>	Pointer to the size of the external work buffer to be used by the DFT functions.



## Description

The function `ippiDFTGetBufSize` is declared in the `ippi.h` file. This function determines the size in bytes of an external memory buffer that can be used by DFT functions with context `pDFTSpec` as a workspace during calculations.

If you want to use external buffering, call this function after `pDFTSpec` initialization. Use the computed `pSize` value as the size in bytes of an external buffer to be allocated by means of, for example, [ippiMalloc](#) functions.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pDFTSpec</code> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <code>pDFTSpec</code> structure is passed.

---

## DFTFwd

*Applies forward discrete Fourier transform to an image.*

---

## Syntax

### Case 1: Not-in-place operation on integer data

```
ippStatus ippiDFTFwd_RToPack_<mod> (const Ipp8u* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, const IppiDFTSpec_R_32s* pDFTSpec,
    int scaleFactor, Ipp8u* pBuffer);
```

Supported values for `mod`:

```
8u32s_C1RSfs
8u32s_C3RSfs
8u32s_C4RSfs
8u32s_AC4RSfs
```

### Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiDFTFwd_RToPack_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,
    Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

### Case 3: Not-in-place operation on complex data

```
IppStatus ippiDFTFwd_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep,
    Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc* pDFTSpec,
    Ipp8u* pBuffer);
```

### Case 4: In-place operation on floating-point data

```
IppStatus ippiDFTFwd_RToPack_<mod>(Ipp32f* pSrcDst, int srcDstStep,
    const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

### Case 5: In-place operation on complex data

```
IppStatus ippiDFTFwd_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep,
    const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.

<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>pDFTSpec</i>	Pointer to the previously initialized DFT context structure.
<i>scaleFactor</i>	The factor for integer result scaling (see <a href="#">Integer Result Scaling</a> ).
<i>pBuffer</i>	Pointer to the external buffer to be used by the DFT functions.

## Description

The function `ippiDFTFwd` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) of the size specified by the function `ippiDFTInitAlloc`.

This function performs a forward DFT on each channel of the source image ROI *pSrc* (*pSrcDst* for in-place flavors) and writes the Fourier coefficients into the corresponding channel of the destination buffer *pDst* (*pSrcDst* for in-place flavors).

The function flavor `ippiDFTFwd_RToPack` that operates on images with real data takes advantage of the symmetry property and stores the output data in [RCPack2D format](#). It supports processing of the 1-, 3-, and 4-channel images. Note that the functions with AC4 descriptor do not process alpha channel.

The function flavor `ippiDFTFwd_CToC` that operates on images with complex data performs no packing of the transform results as no symmetry with respect to frequency domain data is observed in this case. Memory layout of images with complex data follows the same conventions as for real images provided that each pixel value consists of two numbers: imaginary and real part.

The forward DFT functions use the previously initialized *pDFTSpec* context structure to set the mode of calculations and retrieve support data.

The application can also pass the pointer to the external work buffer *pBuffer*. In this case the required buffer size must be determined by calling the [ippiDFTGetBufSize](#) function prior to using the DFT computation functions. If a null pointer is passed, memory allocation will be handled by the `ippiDFTFwd` function internally.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pDFTSpec</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDFTSpec</i> structure is passed.

ippStsMemAllocErr

Indicates an error condition if memory allocation fails.

---

## DFTInv

*Applies an inverse DFT to complex source data and stores results in a destination image.*

---

### Syntax

#### Case 1: Not-in-place operation on integer data

```
IppStatus ippIDFTInv_PackToR_<mod>(const Ipp32s* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, const IppiDFTSpec_R_32s* pDFTSpec,  
    int scaleFactor, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32s8u_C1RSfs  
32s8u_C3RSfs  
32s8u_C4RSfs  
32s8u_AC4RSfs
```

#### Case 2: Not-in-place operation on floating-point data

```
IppStatus ippIDFTInv_PackToR_<mod>(const Ipp32f* pSrc, int srcStep,  
    Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec,  
    Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1R  
32f_C3R  
32f_C4R  
32f_AC4R
```

#### Case 3: Not-in-place operation on complex data

```
IppStatus ippIDFTInv_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep,  
    Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc* pDFTSpec,  
    Ipp8u* pBuffer);
```

#### Case 4: In-place operation on floating-point data

```
IppStatus ippIDFTInv_PackToR_<mod>(Ipp32f* pSrcDst, int srcDstStep,  
    const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

### Case 5: In-place operation on complex data

```
IppStatus ippIDFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep,
    const IppIDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>pDFTSpec</i>	Pointer to the previously initialized DFT context structure.
<i>scaleFactor</i>	The factor for integer result scaling (see <a href="#">Integer Result Scaling</a> ).
<i>pBuffer</i>	Pointer to the external buffer to be used by the DFT functions.

### Description

The function `ippIDFTInv` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) of the size specified by the function `ippIDFTInitAlloc`.

This function performs an inverse DFT on each channel of the input buffer *pSrc* (*pSrcDst* for in-place flavors) and writes the restored image data into the corresponding channel of the output image buffer *pDst* (*pSrcDst* for in-place flavors).

For function flavor `ippIDFTInv_PackToR`, the input buffer must contain data in [RCPack2D format](#).

The inverse DFT functions use the previously initialized *pDFTSpec* context structure to set the mode of calculations and retrieve support data.

The application can also pass the pointer to the external work buffer *pBuffer*. In this case the required buffer size must be determined by calling the [ippiDFTGetBufSize](#) function prior to using the DFT computation functions. If a null pointer is passed, memory allocation will be handled by the *ippiDFTInv* function internally.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pDFTSpec</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDFTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## MulPack

*Multiplies two source images in packed format.*

---

### Syntax

#### Case 1: Not-in-place operation on integer data

```
IppStatus ippiMulPack_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int
    dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

16s_C1RSfs	32s_C1RSfs
16s_C3RSfs	32s_C3RSfs
16s_C4RSfs	32s_C4RSfs
16s_AC4RSfs	32s_AC4RSfs

#### Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiMulPack_<mod>(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

#### Case 3: In-place operation on integer data

```
IppStatus ippiMulPack_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pSrcDst, int dstSrcStep, IppiSize roiSize,
    int scaleFactor);
```

Supported values for *mod*:

16s_C1IRSfs	32s_C1IRSfs
16s_C3IRSfs	32s_C3IRSfs
16s_C4IRSfs	32s_C4IRSfs
16s_AC4IRSfs	32s_AC4IRSfs

**Case 4: In-place operation on floating-point data**

```
IppStatus ippiMulPack_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pSrcDst,  
    int dstSrcStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR  
32f_C3IR  
32f_C4IR  
32f_AC4IR
```

**Parameters**

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the ROI in the source images.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source image buffers.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the second source and destination image ROI for the in-place operation.
<i>dstSrcStep</i>	Distance in bytes between starts of consecutive lines in the second source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling (see <a href="#">Integer Result Scaling</a> ).

**Description**

The function `ippiMulPack` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies corresponding pixel values of two source images, *A* and *B* represented in [RCPack2D format](#) and stores the result into the destination image *C* in packed format also. The multiplying is performed according to the following formulas:

$$\begin{aligned}\text{Re}C &= \text{Re}A * \text{Re}B - \text{Im}A * \text{Im}B; \\ \text{Im}C &= \text{Im}A * \text{Re}B + \text{Im}B * \text{Re}A\end{aligned}$$



Not-in-place flavors multiply pixel values of ROI in the source images *pSrc1* and *pSrc2*, and store result in the *pDst*.

In-place flavors multiply pixel values of ROI in the source images *pSrc* and *pSrcDst*, and store result in the *pSrcDst*.

In case of operations on integer data, the resulting values are scaled by *scaleFactor* (see [Integer Result Scaling](#)).

Note that the functions with AC4 descriptor do not process the alpha channel.

This function can be used in image filtering operations that include FFT transforms.

The [Example 10-3](#) illustrates how the horizontal edges of an image can be detected. First, both the source image and the filter are transformed into the frequency domain by applying the forward FFT function which yields results in packed format. Then, actual filtering is performed through multiplying these packed data on a point by point basis using the function `ippiMulPack`. Finally, filtered data is transformed to the time domain by means of the inverse FFT function.

### Example 10-3 Using `ippiMulPack` Function in Image Filtering

```
IppStatus mulpack( void ) {
    Ipp32f tsrc[64], tflt[64], tdst[64];
    Ipp32f fsrc[64], fflt[64], fdst[64];
    IppiFFTSpec_R_32f *spec;
    const IppiSize roiSize8x8 = { 8, 8 }, roiSize3x3 = { 3, 3 };
    const Ipp32f filter[3*3] = {-1,-1,-1, 0,0,0, 1,1,1};
    ippiSet_32f_C1R( 0, tsrc, 8*sizeof(Ipp32f), roiSize8x8 );
    ippiAddC_32f_C1R( 1, tsrc+8+1, 8*sizeof(Ipp32f), roiSize3x3 );
    ippiSet_32f_C1R( 0, tflt, 8*sizeof(Ipp32f), roiSize8x8 );
    ippiCopy_32f_C1R( filter, 3*sizeof(Ipp32f), tflt, 8*sizeof(Ipp32f),
                     roiSize3x3 );
    ippiFFTInitAlloc_R_32f( &spec, 3, 3, IPP_FFT_DIV_INV_BY_N,
                           ippAlgHintAccurate );
    ippiFFTFwd_RToPack_32f_C1R( tsrc, 8*sizeof(Ipp32f), fsrc,
                                8*sizeof(Ipp32f), spec, 0 );
    ippiFFTFwd_RToPack_32f_C1R( tflt, 8*sizeof(Ipp32f), fflt,
                                8*sizeof(Ipp32f), spec, 0 );
    ippiMulPack_32f_C1R( fsrc, 8*sizeof(Ipp32f), fflt, 8*sizeof(Ipp32f),
                        fdst, 8*4, roiSize8x8 );
    ippiFFTInv_PackToR_32f_C1R( fdst, 8*sizeof(Ipp32f), tdst,
                                8*sizeof(Ipp32f), spec, 0 );
    ippiFFTFree_R_32f( spec );
    return 0;
}
```

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

---

## MulPackConj

*Multiplies a source image by the complex conjugate image with data in packed format and stores the result in the destination buffer in the packed format.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippMulPackConj_<mod>(const Ipp32f* pSrc1, int src1Step, const
    Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize
    roiSize);
```

Supported values for *mod*:

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

#### Case 2: In-place operation

```
IppStatus ippMulPackConj_<mod>(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR
```

## Parameters

<i>pSrc, pSrc1, pSrc2</i>	Pointers to the ROI in the source images.
<i>srcStep, src1Step, src2Step</i>	Distances in bytes between starts of consecutive lines in the source image buffers.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the second source and destination image ROI for the in-place operation.
<i>dstSrcStep</i>	Distance in bytes between starts of consecutive lines in the second source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiMulPackConj` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function multiplies pixel values of the source image  $A$  by the corresponding pixel values of the complex conjugate image  $A^*$ , represented in [RCPack2D format](#). The result of the operation is written into the destination buffer in packed format also.

Not-in-place flavors multiply pixel values of ROI in the source images *pSrc1* and *pSrc2*, and store result in the *pDst*.

In-place flavors multiply pixel values of ROI in the source images *pSrc* and *pSrcDst*, and store result in the *pSrcDst*.

## Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippiStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

`ippStsStepErr` Indicates an error condition if any of the specified buffer step values is zero or negative.

---

## Magnitude

*Computes magnitude of elements of a complex data image.*

---

### Syntax

```
IppStatus ippiMagnitude_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

`32fc32f_C1R`     `32fc32f_C3R`

```
IppStatus ippiMagnitude_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

`16sc16s_C1RSfs`   `16sc16s_C3RSfs`  
`32sc32s_C1RSfs`   `32sc32s_C3RSfs`

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling (see <a href="#">Integer Result Scaling</a> ).

### Description

The function `ippiMagnitude` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes magnitude of elements of the source image *pSrc* given in complex data format, and stores results in the destination image *pDst*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of images is less than or equal to zero.

---

## MagnitudePack

*Computes magnitude of elements  
of an image in packed format.*

---

### Syntax

```
IppStatus ippMagnitudePack_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod*:

```
32f_C1R      32f_C3R
```

```
IppStatus ippMagnitudePack_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    int scaleFactor);
```

Supported values for *mod*:

```
16s_C1RSfs   16s_C3RSfs
32s_C1RSfs   32s_C3RSfs
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiMagnitudePack` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes magnitude of elements of the source image *pSrc* given in [RCPack2D format](#) and stores results in the destination image *pDst*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of images is less than or equal to zero.

---

## Phase

*Computes the phase of elements  
of a complex data image.*

---

## Syntax

```
ippiStatus ippiPhase_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
32fc32f_C1R    32fc32f_C3R
```

```
ippiStatus ippiPhase_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
    Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for *mod*:

16sc16s_C1RSfs	32sc32s_C1RSfs
16sc16s_C3RSfs	32sc32s_C3RSfs

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling (see <a href="#">Integer Result Scaling</a> ).

## Description

The function `ippiPhase` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the phase in radians of elements of a source image *pSrc* given in complex data format, and stores results in the destination image *pDst*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of images is less than or equal to zero.

---

## PhasePack

*Computes the phase of elements  
of an image in packed format.*

---

### Syntax

```
IppStatus ippiPhasePack_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize);
```

Supported values for *mod*:

32f\_C1R            32f\_C3R

```
IppStatus ippiPhasePack_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int scaleFactor);
```

Supported values for *mod*:

16s\_C1RSfs            32s\_C1RSfs  
16s\_C3RSfs            32s\_C3RSfs

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>scaleFactor</i>	The factor for integer result scaling (see <a href="#">Integer Result Scaling</a> ).

### Description

The function `ippiPhasePack` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the phase of elements of a source image *pSrc* given in [RCPack2D format](#) and stores results in the destination image *pDst*.



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of images is less than or equal to zero.

## PolarToCart

*Converts an image in the polar coordinate form to Cartesian coordinate form.*

### Syntax

```
IppStatus ippipolarToCart_32f32fc_P2C1R(const Ipp32f* pSrcMagn,
    const Ipp32f* pSrcPhase, int srcStep, Ipp32fc* pDst, int dstStep,
    IppiSize roiSize);
```

### Parameters

<i>pSrcMagn</i>	Pointer to the ROI in the source image plane containing magnitudes.
<i>pSrcPhase</i>	Pointer to the ROI in source image plane containing phase values.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source buffers.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>roiSize</i>	Size of the source and destination image ROI.

### Description

The function `ippipolarToCart` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a two-plane source image in the polar coordinate form to the one-channel image in complex-data format (in Cartesian coordinate form) *pDst*. The first plane of the source image *pSrcMagn* contains magnitude values, the second plane of the source image *pSrcPhase* contains phase values in radians.

### Return Values

<i>ippStsNoErr</i>	Indicates no error. Any other value indicates an error or a warning.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one of the specified pointers is NULL.
<i>ippStsStepErr</i>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>srcSize</i> has a field with value less than 1.

---

## PackToCplxExtend

*Converts an image in packed format to a complex data image.*

---

### Syntax

```
IppStatus ippiPackToCplxExtend_32s32sc_C1R(const Ipp32s* pSrc,
      IppiSize srcRoiSize, int srcStep, Ipp32sc* pDst, int dstStep);
IppStatus ippiPackToCplxExtend_32f32fc_C1R(const Ipp32f* pSrc,
      IppiSize srcRoiSize, int srcStep, Ipp32fc* pDst, int dstStep);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcRoiSize</i>	Size in pixels of the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source buffer.
<i>pDst</i>	Pointer to the destination image buffer.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.

## Description

The function `ippiPackToCplxExtend` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the source image `pSrc` in [RCPack2D format](#) to complex data format and stores the results in `pDst`, which is a matrix with complete set of the Fourier coefficients. Note that if the `pSrc` in RCPack2D format is a real array of dimensions (N×M), then the `pDst` is a real array of dimensions (2×N×M). This should be taken into account when allocating memory for the function operation.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcSize</code> has field with zero or negative value.

## Windowing Functions

This section describes Intel IPP windowing functions used in image processing. A window is a mathematical function by which pixel values are multiplied to prepare an image for the subsequent analysis. This procedure is often called ‘windowing’. In fact, a window function approaches zero towards the edges of the image avoiding strong distortions of spectral densities in the Fourier domain.

The Intel IPP provides two following types of window functions:

- Bartlett window function
- Hamming window function

These functions generate the window samples and applied them to the specified image. To obtain the window samples themselves, you should apply the desired function to the image with all pixel values set to 1.0. As the windowing operation is very time consuming, it may be useful if you want to apply the same window to the multiple images. In this case use one of the image multiplication functions (`ippiMul`) to multiply the pixel values of the image by the window samples.

## WinBartlett, WinBartlettSep

*Applies Bartlett window function to the image.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiWinBartlett_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1R                  32f\_C1R

```
IppStatus ippiWinBartlettSep_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1R                  32f\_C1R

#### Case 2: In-place operation

```
IppStatus ippiWinBartlett_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1IR                32f\_C1IR

```
IppStatus ippiWinBartlettSep_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1IR                32f\_C1IR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiWinBartlett` and `ippiWinBartlettSep` are declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

These functions compute the Bartlett (triangle) window samples, multiply pixel values of the source image *pSrc* (*pSrcDst* for in-place flavors) with these samples, and store results in the destination image *pDst* (*pSrcDst* for in-place flavors).

The Bartlett window function for one-dimensional case with *M* elements is defined as follows:

$$w_{\text{bartlett}}(i) = \begin{cases} \frac{2i}{M-1}, & 0 \leq i \leq \frac{M-1}{2} \\ 2 - \frac{2i}{M-1}, & \frac{M-1}{2} < i \leq M-1 \end{cases}$$

**ippiWinBartlettSep.** This flavor applies the window function successively to the rows and then to the columns of the image.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of images is less than or equal to zero.

---

## WinHamming, WinHammingSep

*Applies Hamming window function to the image.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiWinHamming_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1R                  32f\_C1R

```
IppStatus ippiWinHammingSep_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1R                  32f\_C1R

#### Case 2: In-place operation

```
IppStatus ippiWinHamming_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,  
    IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1R                  32f\_C1R

```
IppStatus ippiWinHammingSep_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,  
    IppiSize roiSize);
```

Supported values for *mod*:

8u\_C1R                  32f\_C1R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiWinHamming` and `ippiWinHammingSep` are declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

These functions compute the Bartlett (triangle) window samples, multiply pixel values of the source image *pSrc* (*pSrcDst* for in-place flavors) with these samples, and store results in the destination image *pDst* (*pSrcDst* for in-place flavors).

The Hamming window function for one-dimensional case with *M* elements is defined as follows:

$$w_{\text{hamming}}(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{len-1}\right)$$

**ippiWinBartlettSep.** This flavor applies the window function successively to the rows and then to the columns of the image.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of images is less than or equal to zero.

## Discrete Cosine Transforms

Discrete Cosine Transform (DCT) of a real 2D image yields output results that are also real, which eliminates the need to use packed format for storing the transformed data. However, forward and inverse DCT functions `ippiDCTFwd` and `ippiDCTInv` need different context data structures to be initialized and filled in prior to their use. Consequently, the required workspace buffer size is different for these functions. In case of using an external buffer, its size must be determined by previously calling the respective support function. DCT functions that use context structures implement the modified computation algorithm proposed in [\[Rao90\]](#).

The DCT functions `ippiDCT8x8Fwd` and `ippiDCT8x8Inv` working on a fixed 8x8 image buffer need no context data or external workspace buffers. Functions `ippiDCT8x8Inv` meet IEEE-1180 standard requirements (see [\[IEEE\]](#)).

Intel IPP Discrete Cosine Transform functions working on a fixed 8x8 image buffer use Feig and Winograd algorithm ([\[Feig92\]](#)) modified for taking advantage of SIMD instructions. For details on algorithms used in DCT transforms and for more references, see [\[AP922\]](#).

---

### DCTFwdInitAlloc

*Allocates memory and fills in context data needed for the forward DCT function to operate.*

---

#### Syntax

```
IppStatus ippiDCTFwdInitAlloc_32f (IppiDCTFwdSpec_32f** pDCTSpec,  
    IppiSize roiSize, IppHintAlgorithm hint);
```

#### Parameters

<i>pDFTSpec</i>	Pointer to the forward DCT context structure being initialized.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>hint</i>	Option to select the algorithmic implementation of the transform function.



## Description

The function `ippiDCTFwdInitAlloc` is declared in the `ippi.h` file. This function allocates memory and initializes the context structure `pDCTSpec` needed to compute the forward DCT of a two-dimensional image data. The [ippiDCTFwd](#) function called with the pointer to the initialized `pDCTSpec` structure as an argument will compute the forward discrete cosine transform for points in the ROI of size `roiSize`, using computation algorithm indicated by `hint` parameter (see [Table 10-2](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pDCTSpec</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## DCTInvInitAlloc

*Allocates memory and fills in context data needed for the inverse DCT function to operate.*

---

## Syntax

```
IppStatus ippiDCTInvInitAlloc_32f (IppiDCTInvSpec_32f** pDCTSpec,  
    IppiSize roiSize, IppHintAlgorithm hint);
```

## Parameters

<code>pDFTSpec</code>	Pointer to the inverse DCT context structure being initialized.
<code>roiSize</code>	Size of the source and destination ROI in pixels.
<code>hint</code>	Option to select the algorithmic implementation of the transform function.

## Description

The function `ippiDCTInvInitAlloc` is declared in the `ippi.h` file. This function allocates memory and initializes the context structure `pDCTSpec` needed to compute the inverse DCT of a two-dimensional image data. The `ippiDCTInv` function called with the pointer to the initialized `pDCTSpec` structure as an argument will compute the inverse discrete cosine transform for points in the ROI of size `roiSize`, using computation algorithm indicated by `hint` (see [Table 10-2](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pDCTSpec</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## DCTFwdFree

*Deallocates memory used by the forward DCT context structure.*

---

## Syntax

```
IppStatus ippiDCTFwdFree_32f (IppiDCTFwdSpec_32f** pDCTSpec);
```

## Parameters

`pDCTSpec`                      Pointer to the forward DCT context structure that has to be released.

## Description

The function `ippiDCTFwdFree` is declared in the `ippi.h` file. This function releases memory used by the previously initialized forward DCT context structure `pDCTSpec`. Call this function after your application program has finished computing the forward discrete cosine transforms.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pDCTSpec</code> pointer is NULL.

`ippStsContextMatchErr` Indicates an error condition if a pointer to an invalid `pDCTSpec` structure is passed.

---

## DCTInvFree

*Deallocates memory used by the inverse DCT context structure.*

---

### Syntax

```
IppStatus ippIDCTInvFree_32f (IppiDCTInvSpec_32f** pDCTSpec);
```

### Parameters

`pDCTSpec` Pointer to the inverse DCT context structure that has to be released.

### Description

The function `ippIDCTInvFree` is declared in the `ippi.h` file. This function releases memory used by the previously initialized inverse DCT context structure `pDCTSpec`. Call this function after your application program has finished computing the inverse discrete cosine transforms.

### Return Values

`ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.

`ippStsNullPtrErr` Indicates an error condition if `pDCTSpec` pointer is `NULL`.

`ippStsContextMatchErr` Indicates an error condition if a pointer to an invalid `pDCTSpec` structure is passed.

---

## DCTFwdGetBufSize

*Determines the size of work buffer can be used by the forward DCT function to operate.*

---

### Syntax

```
IppStatus ippIDCTFwdGetBufSize_32f (IppiDCTFwdSpec_32f* pDCTSpec,  
                                     int* pSize);
```

### Parameters

<i>pDCTSpec</i>	Pointer to the previously initialized forward DCT context structure.
<i>pSize</i>	Pointer to the size of the external work buffer to be used by the forward DCT function.

### Description

The function `ippiDCTFwdGetBufSize` is declared in the `ippi.h` file. This function determines the size in bytes of an external memory buffer that can be used by the forward DCT function with context *pDCTSpec* as a workspace during calculations.

If you want to use external buffering, call this function after *pDCTSpec* initialization.

Use the computed *pSize* value as the size in bytes of an external buffer to be allocated by means of, for example, `ippiMalloc` functions.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pDCTSpec</i> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDCTSpec</i> structure is passed.

---

## DCTInvGetBufSize

*Determines the size of work buffer that can be used by the inverse DCT function to operate.*

---

### Syntax

```
IppStatus ippiDCTInvGetBufSize_32f (IppiDCTInvSpec_32f* pDCTSpec,
                                     int* pSize);
```

### Parameters

<i>pDCTSpec</i>	Pointer to the previously initialized inverse DCT context structure.
<i>pSize</i>	Pointer to the size of the external work buffer to be used by the inverse DCT function.

## Description

The function `ippiDCTInvGetBufSize` is declared in the `ippi.h` file. This function determines the size in bytes of an external memory buffer that can be used by the inverse DCT function with context `pDCTSpec` as a workspace during calculations.

If you want to use external buffering, call this function after `pDCTSpec` initialization. Use the computed `pSize` value as the size in bytes of an external buffer to be allocated by means of, for example, `ippiMalloc` functions.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pDCTSpec</code> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <code>pDCTSpec</code> structure is passed.

---

## DCTFwd

*Applies a forward discrete cosine transform to an image.*

---

## Syntax

```
IppStatus ippiDCTFwd_<mod> (const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiDCTFwdSpec_32f* pDCTSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

```
32f_C1R  
32f_C3R  
32f_C4R  
32f_AC4R
```

## Parameters

*pSrc*                      Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pDCTSpec</i>	Pointer to the previously initialized forward DCT context structure.
<i>pBuffer</i>	Pointer to the external buffer to be used by the forward DCT function.

## Description

The function `ippiDCTFwd` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) of the size that is specified by the function [ippiDCTFwdInitAlloc](#).

This function performs a forward DCT on each channel of the source image *pSrc* and writes the result into the corresponding channel of the destination image buffer *pDst*. Note that the function flavor with `AC4` descriptor does not process alpha channel.

This function uses the previously initialized *pDCTSpec* context structure to set the mode of calculations and retrieve support data.

The application can also pass the pointer to the external work buffer *pBuffer*.

In this case the required buffer size must be determined by calling the [ippiDCTFwdGetBufSize](#) function prior to using the forward DCT computation function. If a null pointer is passed, memory allocation will be handled by the `ippiDCTFwd` function internally.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pDCTSpec</i> pointer is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDCTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## DCTInv

*Applies an inverse discrete cosine transform to an image.*

### Syntax

```
IppStatus ippiDCTInv_<mod> (const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, const IppiDCTInvSpec_32f* pDCTSpec, Ipp8u*
    pBuffer);
```

Supported values for *mod* :

```
32f_C1R
32f_C3R
32f_C4R
32f_AC4R
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pDCTSpec</i>	Pointer to the previously initialized inverse DCT context structure.
<i>pBuffer</i>	Pointer to the external buffer to be used by the inverse DCT function.

### Description

The function `ippiDCTInv` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) of the size that is specified by the function `ippiDCTInvInitAlloc`.

This function performs an inverse DCT on each channel of the input image *pSrc* and writes the result into the corresponding channel of the output image buffer *pDst*. Note that the function flavor with AC4 descriptor does not process alpha channel.

This function uses the previously initialized *pDCTSpec* context structure to set the mode of calculations and retrieve support data.

The application can also pass the pointer to the external work buffer *pBuffer*. In this case the required buffer size must be determined by calling the [ippiDCTInvGetBufSize](#) function prior to using the inverse DCT computation function. If a null pointer is passed, memory allocation will be handled by the `ippiDCTInv` function internally.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pDCTSpec</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> value is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid <i>pDCTSpec</i> structure is passed.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## DCT8x8Fwd

*Performs a forward DCT on a 2D buffer of 8x8 size.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippiDCT8x8Fwd_<mod>(const Ipp<datatype>* pSrc,
                                Ipp<datatype>* pDst);
```

Supported values for *mod*:

```
16s_C1      32f_C1
```

#### Case 2: Not-in-place operation with ROI

```
IppStatus ippiDCT8x8Fwd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
                                Ipp<dstDatatype>* pDst);
```

Supported values for *mod*:

```
16s_C1R      8u16s_C1R
```



**Case 3: In-place operation**

```
IppStatus ippiDCT8x8Fwd_<mod>(Ipp<datatype>* pSrcDst );
```

Supported values for *mod*:

```
16s_C1I      32f_C1I
```

**Parameters**

<i>pSrc</i>	Pointer to the source image buffer.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer for operations with ROI.
<i>pDst</i>	Pointer to the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination image for in-place operations.

**Description**

The function `ippiDCT8x8Fwd` is declared in the `ippi.h` file. Some flavors operate with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the forward discrete cosine transform of short integer or floating-point data in a 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

The code [Example 10-4](#) illustrates the use of `ippiDCT8x8Fwd` function.

**Example 10-4 Forward DCT of 8x8 Size**

```
IppStatus dct16s( void ) {
    Ipp16s x[64] = {0};
    IppiSize roi = {8,8};
    int i;
    for( i=0; i<8; ++i ) {
        ippiSet_16s_C1R( (Ipp16s)i, x+8*i+i, 8*sizeof(Ipp16s), roi );
        --roi.width;
        --roi.height;
    }
    return ippiDCT8x8Fwd_16s_C1I( x );
}
```

### Example 10-4 Forward DCT of 8x8 Size (continued)

---

The destination image *x* contains:

```

18 -9 -2 -1  0  0  0  0
-9  7  0  0  0  0  0  0
-2  0  2  0  0  0  0  0
-1  0  0  1  0  0  0  0
 0  0  0  0  1  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0

```

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is zero or negative.

---

## DCT8x8Inv

*Performs an inverse DCT  
on a 2D buffer of 8x8 size.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippIDCT8x8Inv_<mod>(const Ipp<datatype>* pSrc,
                               Ipp<datatype>* pDst);
```

Supported values for *mod*:

```
16s_C1      32f_C1
```

#### Case 2: Not-in-place operation with ROI

```
IppStatus ippIDCT8x8Inv_<mod>(const Ipp<srcDatatype>* pSrc,
                               Ipp<dstDatatype>* pDst, int dstStep);
```

Supported values for *mod*:

16s\_C1R      16s8u\_C1R

### Case 3: In-place operation

```
IppStatus ippiDCT8x8Inv_<mod>(Ipp<datatype>* pSrcDst);
```

Supported values for *mod*:

16s\_C1I      32f\_C1I

### Parameters

<i>pSrc</i>	Pointer to the source image.
<i>pDst</i>	Pointer to the destination buffer.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination buffer for operations with ROI.
<i>pSrcDst</i>	Pointer to the source and destination image for in-place operations.

### Description

The function `ippiDCT8x8Inv` is declared in the `ippi.h` file. Some flavors operate with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the inverse discrete cosine transform of short integer or floating-point data in a 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pDst</i> , or <i>pSrcDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> value is zero or negative.

---

## DCT8x8FwdLS

*Performs a forward DCT on a 2D buffer of 8x8 size with prior data conversion and level shift.*

---

### Syntax

```
IppStatus ippiDCT8x8FwdLS_8u16s_C1R(const Ipp8u* pSrc, int srcStep,  
    Ipp16s* pDst, Ipp16s addVal);
```

### Parameters

<i>pSrc</i>	Pointer to the source image buffer.
<i>pDst</i>	Pointer to the destination buffer.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>addVal</i>	The level shift value.

### Description

The function `ippiDCT8x8FwdLS` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) that is a 2D buffer of 8x8 size in this case, thus there is no need to specify its size.

This function first converts data in the buffer *pSrc* from unsigned `Ipp8u` type to the signed `Ipp16s` type and then performs level shift operation by adding the constant value *addVal* to each sample. After that, the function performs the forward discrete cosine transform of the modified data. The result is stored in *pDst*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is zero or negative.

---

## DCT8x8InvLSClip

*Performs an inverse DCT on a 2D buffer of 8x8 size with further data conversion and level shift.*

---

### Syntax

```
IppStatus ippiDCT8x8InvLSClip_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst,
    int dstStep, Ipp16s addVal, Ipp8u clipDown, Ipp8u clipUp);
```

### Parameters

<i>pSrc</i>	Pointer to the source image buffer.
<i>pDst</i>	Pointer to the destination image buffer.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>addVal</i>	The level shift value.
<i>clipDown</i>	The lower bound for the range of output values.
<i>clipUp</i>	The upper bound for the range of output values.

### Description

The function `ippiDCT8x8InvLSClip` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) that is a 2D buffer of 8x8 size in this case, thus there is no need to specify its size.

This function performs the inverse discrete cosine transform of the buffer *pSrc*. After completing the DCT, this function performs level shift operation by adding the constant value *addVal* to each sample. Finally, the function converts data from the signed `Ipp16s` type to the unsigned `Ipp8u` type. The output data are clipped to the range `[clipDown .. clipUp]`. The result is stored in the destination buffer *pDst*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> value is zero or negative.

---

## DCT8x8Inv\_2x2, DCT8x8Inv\_4x4

*Perform an inverse DCT on a top left quadrant of size 2x2 or 4x4 in the 2D buffer of size 8x8.*

---

### Syntax

```
IppStatus ippiDCT8x8Inv_2x2_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);  
IppStatus ippiDCT8x8Inv_4x4_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);  
IppStatus ippiDCT8x8Inv_2x2_16s_C1I(Ipp16s* pSrcDst);  
IppStatus ippiDCT8x8Inv_4x4_16s_C1I(Ipp16s* pSrcDst);
```

### Parameters

<i>pSrc</i>	Pointer to the source image.
<i>pDst</i>	Pointer to the destination buffer.
<i>pSrcDst</i>	Pointer to the source and destination buffer for in-place operations.

### Description

The functions `ippiDCT8x8Inv_2x2` and `ippiDCT8x8Inv_4x4` are declared in the `ippi.h` file. These functions compute the inverse discrete cosine transform of non-zero elements in the top left quadrant of size 2x2 or 4x4 in the 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

# Image Statistics Functions

11

This chapter describes the Intel Integrated Performance Primitives functions that can be used to compute the following statistical parameters of an image:

- sum, integrals, mean and standard deviation of pixel values
- intensity histogram of pixel values
- minimum and maximum pixel values
- spatial and central moments of order 0 to 3
- the infinity,  $L1$ , and  $L2$  norms of the image pixel values and of the differences between pixel values of two images
- relative error values for the infinity,  $L1$ , and  $L2$  norms of differences between pixel values of two images
- universal image quality index
- proximity measures of an image and a template (another image).

[Table 11-1](#) lists the image statistics functions.:

**Table 11-1 Image Statistics Functions**

Function Base Name	Operation
<a href="#">Sum</a>	Computes the sum of image pixel values
<a href="#">Integral</a>	Transforms an image to the integral representation
<a href="#">SqrIntegral</a>	Transforms an image to integral and integral of pixel squares representations
<a href="#">TiltedIntegral</a>	Transforms an image to the tilted integral representation
<a href="#">TiltedSqrIntegral</a>	Transforms an image to tilted integral and tilted integral of pixel squares representations

**Table 11-1 Image Statistics Functions (continued)**

Function Base Name	Operation
<a href="#">Mean</a>	Computes the mean of image pixel values
<a href="#">Mean StdDev</a>	Computes the mean and standard deviation of image pixel values
<a href="#">RectStdDev</a>	Computes the standard deviation of the integral images.
<a href="#">TiltedRectStdDev</a>	Computes the standard deviation of the tilted integral images.
<a href="#">HistogramRange</a>	Computes the intensity histogram of an image
<a href="#">HistogramEven</a>	Computes the intensity histogram of an image with equal bins
<a href="#">CountInRange</a>	Counts the number of pixels within the given intensity range
<b>Minimum and Maximum Operations</b>	
<a href="#">Min</a>	Computes the minimum of image pixel values
<a href="#">MinIndx</a>	Computes the minimum of image pixel values, and retrieves the coordinates of pixels with minimal intensity values
<a href="#">Max</a>	Computes the maximum of image pixel values
<a href="#">MaxIndx</a>	Computes the maximum of image pixel values, and retrieves the coordinates of pixels with maximal intensity values
<a href="#">MinMax</a>	Computes the minimum and maximum of image pixel values
<a href="#">MinMaxIndx</a>	Calculates minimum and maximum pixel values and their indexes
<b>Moments</b>	
<a href="#">MomentInitAlloc</a>	Allocates memory and initializes the moment context structure
<a href="#">MomentFree</a>	Deallocates the previously initialized structure that stores image moments
<a href="#">MomentGetStateSize</a>	Computes the size of the external buffer for the moment context structure
<a href="#">MomentInit</a>	Initializes the moment context structure
<a href="#">Moments</a>	Computes all image moments of order 0 to 3 and Hu moment invariants
<a href="#">GetSpatialMoment</a>	Retrieves image spatial moment computed by <code>ippiMoments</code>



**Table 11-1** Image Statistics Functions (continued)

Function Base Name	Operation
<a href="#"><u>GetCentralMoment</u></a>	Retrieves image central moment computed by <code>ippiMoments</code>
<a href="#"><u>GetNormalizedSpatialMoment</u></a>	Retrieves the normalized value of the image spatial moment computed by <code>ippiMoments</code>
<a href="#"><u>GetNormalizedCentralMoment</u></a>	Retrieves the normalized value of the image central moment computed by <code>ippiMoments</code>
<a href="#"><u>GetHuMoments</u></a>	Retrieves image Hu moment invariants computed by <code>ippiMoments</code>
<b>Norms</b>	
<a href="#"><u>Norm_Inf</u></a>	Computes the infinity norm of image pixel values
<a href="#"><u>Norm_L1</u></a>	Computes the L1- norm of image pixel values
<a href="#"><u>Norm_L2</u></a>	Computes the L2- norm of image pixel values
<a href="#"><u>NormDiff_Inf</u></a>	Computes the infinity norm of differences between pixel values of two images
<a href="#"><u>NormDiff_L1</u></a>	Computes the L1- norm of differences between pixel values of two images
<a href="#"><u>NormDiff_L2</u></a>	Computes the L2- norm of differences between pixel values of two images
<a href="#"><u>NormRel_Inf</u></a>	Computes the relative error for the infinity norm of differences between pixel values of two images
<a href="#"><u>NormRel_L1</u></a>	Computes the relative error for the L1- norm of differences between pixel values of two images.
<a href="#"><u>NormRel_L2</u></a>	Computes the relative error for the L2- norm of differences between pixel values of two images.
<b>Image Quality Index</b>	
<a href="#"><u>QualityIndex</u></a>	Computes the universal image quality index
<b>Proximity Measures</b>	
<a href="#"><u>SqrDistanceFull_Norm</u></a>	Computes normalized full Euclidean distance between an image and a template.
<a href="#"><u>SqrDistanceSame_Norm</u></a>	Computes normalized Euclidean distance between an image and a template.
<a href="#"><u>SqrDistanceValid_Norm</u></a>	Computes normalized valid Euclidean distance between an image and a template.
<a href="#"><u>CrossCorrFull_Norm</u></a>	Computes normalized full cross-correlation between an image and a template.

**Table 11-1      Image Statistics Functions (continued)**

Function Base Name	Operation
<a href="#">CrossCorrSame_Norm</a>	Computes normalized cross-correlation between an image and a template.
<a href="#">CrossCorrValid_Norm</a>	Computes normalized valid cross-correlation between an image and a template.
<a href="#">CrossCorrFull_NormLevel</a>	Computes normalized full correlation coefficients between an image and a template.
<a href="#">CrossCorrSame_NormLevel</a>	Computes normalized correlation coefficients between an image and a template.
<a href="#">CrossCorrValid_NormLevel</a>	Computes normalized valid correlation coefficients between an image and a template.

## Sum

*Computes the sum of image pixel values.*

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pSum);
```

Supported values for *mod*:

```
8u_C1R      16s_C1R
```

#### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiSum_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pSum, IppHintAlgorithm hint);
```

#### Case 3: Operation on multi-channel integer data

```
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f sum[3]);
```

Supported values for *mod* :

```
8u_C3R      16s_C3R
8u_AC4R      16s_AC4R
```

```
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f sum[4]);
```

Supported values for *mod* :

```
8u_C4R      16s_C4R
```

#### Case 4: Operation on multi-channel floating-point data

```
IppStatus ippiSum_<mod>(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f sum[3], IppHintAlgorithm hint);
```

Supported values for *mod* :

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiSum_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f sum[4], IppHintAlgorithm hint);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pSum</i>	Pointer to the computed sum of pixel values.
<i>sum</i>	Array containing computed sums of channel values of pixels in the source buffer.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The function `ippiSum` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the sum of pixel values *pSum* for the source image *pSrc* using algorithm indicated by the *hint* argument (see [Table 11-3](#)). In case of a multi-channel image, the sum is computed over each channel and stored in the array *sum*.

The following code example demonstrates the use of `ippiSum` function:

### Example 11-1 Summing Up Pixel Values of an Image

---

```
IppStatus sum( void ) {  
    Ipp64f sum;  
    Ipp8u x[5*4];  
    IppiSize roi = {5,4};  
    ippiSet_8u_C1R( 1, x, 5, roi );  
    return ippiSum_8u_C1R( x, 5, roi, &sum);  
}
```

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pSum</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

## Integral

*Transforms an image to the integral representation.*

### Syntax

```
IppStatus ippiIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst,
    int dstStep, IppiSize roiSize, Ipp32s val);
IppStatus ippiIntegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst,
    int dstStep, IppiSize roiSize, Ipp32f val);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination integral image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of source and destination image ROI in pixels.
<i>val</i>	The value to add to <i>pDst</i> image pixels

### Description

The function `ippiIntegral` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function transform a source image *pSrc* to the integral image *pDst*. Pixel values of the destination image *pDst* are computed using pixel values of the source image *pSrc* and the specified value *val* in accordance with the following formula:

$$pDst[i, j] = val + \sum_{k < i} \sum_{l < j} pSrc[k, l]$$

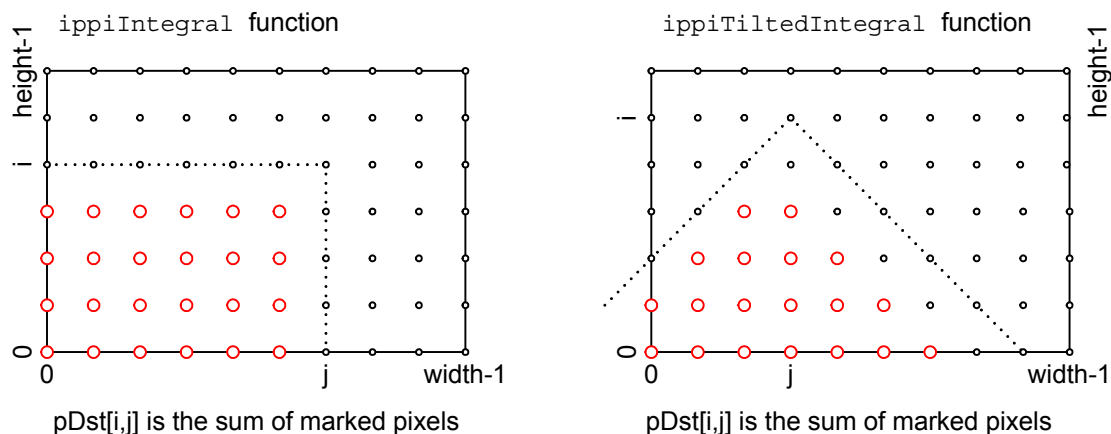
where *i, j* are coordinates of the destination image pixels (see [Figure 11-1](#)) varying in the range *i* = 1, ..., *roiSize.height*, *j* = 0, ..., *roiSize.width*. Pixel values of zero row and column of *pDst* (*i*=0) is set to *val*.

The size of the destination images is (*roiSize.width* + 1) x (*roiSize.height* + 1).

[Figure 11-1](#) shows what pixels (red circles) of the source image are used in computation new pixel values in the  $i, j$  coordinates.

For large images the result of summation can exceed the upper bound of the output data type. [Table 11-2](#) lists the maximum image size for different function flavors and values.

**Figure 11-1 Operation of the Integral and TiltedIntegral functions**



**Table 11-2 Maximum Image Size for Integral Functions**

Function Flavor	Value val	Maximum Image Size
ippiIntegral_8u32s_C1R	0	$(2^{31}-1)/255$
	$-2^{31}$	$2^{32}/255$
ippiIntegral_8u32f_C1R	0	$2^{24}/255$
	$-2^{24}$	$(2^{25}+1)/255$

## Return Values

<code>ippStsNoEr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than ( <i>roiSize.width</i> +1) * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one <i>dstStep</i> is not divisible by 4.

---

## SqrIntegral

*Transforms an image to integral and integral of pixel squares representations.*

---

### Syntax

```
IppStatus ippISqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, Ipp32s* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val,
    Ipp32s valSqr);

IppStatus ippISqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val,
    Ipp64f valSqr);

IppStatus ippISqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32f val,
    Ipp64f valSqr);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination integral image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSqr</i>	Pointer to the ROI of the destination integral image of pixel squares.

<i>sqrStep</i>	Distance in bytes between starts of consecutive lines in the destination integral image of pixel squares.
<i>roiSize</i>	Size of source image ROI in pixels.
<i>val</i>	The value to add to <i>pDst</i> image pixels.
<i>valSqr</i>	The value to add to <i>pSqr</i> image pixels

## Description

The function `ippiSqrIntegral` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function builds two destination image: integral image *pDst* and integral image of pixel squares *pSqr*.

Pixel values of *pDst* are computed using pixel values of the source image *pSrc* and the specified value *val* in accordance with the following formula:

$$pDst[i, j] = val + \sum_{k < i} \sum_{l < j} pSrc[k, l]$$

Pixel values of *pSqr* are computed using pixel values of the source image *pSrc* and the specified value *valSqr* in accordance with the following formula:

$$pSqr[i, j] = valSqr + \sum_{k < i} \sum_{l < j} pSrc[k, l]^2$$

where *i, j* are coordinates of the destination image pixels (see [Figure 11-1](#)) varying in the range *i* = 1, ..., *roiSize.height*, *j* = 0, ..., *roiSize.width*. Pixel values of zero row and column are set to *val* for *pDst*, and to *valSqr* for *pSqr*. The size of both destination images is (*roiSize.width* + 1) x (*roiSize.height* + 1).

[Figure 11-1](#) shows what pixels (red circles) of the source image are used in computation new pixel values in the *i, j* coordinates.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is NULL.



---

<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> , or <code>dstStep</code> or <code>sqrStep</code> is less than <code>(roiSize.width+1) * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if <code>dstStep</code> is not divisible by 4, or <code>sqrStep</code> is not divisible by 8.

---

## TiltedIntegral

*Transforms an image to the tilted integral representation.*

---

### Syntax

```
IppStatus ippiTiltedIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s*
    pDst, int dstStep, IppiSize roiSize, Ipp32s val);
IppStatus ippiTiltedIntegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pDst, int dstStep, IppiSize roiSize, Ipp32f val);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the ROI in the destination integral image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of source image ROI in pixels.
<code>val</code>	The value to add to pDst image pixels

## Description

The function `ippiTiltedIntegral` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function transform a source image `pSrc` to the tilted integral image `pDst`. Pixel values of the destination image `pDst` are computed using pixel values of the source image `pSrc` and the specified value `val` in accordance with the following formula:

$$pDst[i, j] = val + \sum_{\substack{k+1 \leq i+j-2 \\ k-1 \leq i-j+1}} pSrc[k, l]$$

where  $i, j$  are coordinates of the destination image pixels (see [Figure 11-1](#)) varying in the range  $i = 2, \dots, roiSize.height + 1$ ,  $j = 0, \dots, roiSize.width + 1$ . Pixel values of rows 0 and 1 of the destination image `pDst` ( $i=0$ ) is set to `val`.

The size of the destination images is  $(roiSize.width + 2) \times (roiSize.height + 2)$ .

[Figure 11-1](#) shows what pixels (red circles) of the source image are used in computation new pixel values in the  $i, j$  coordinates.

## Return Values

<code>ippStsNoEr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pSrc</code> or <code>pDst</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> , or <code>dstStep</code> is less than $(roiSize.width + 2) * <pixelSize>$ .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one <code>dstStep</code> is not divisible by 4.

## TiltedSqrIntegral

*Transforms an image to tilted integral and tilted integral of pixel squares representations.*

### Syntax

```
IppStatus ippiTiltedSqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, Ipp32s* pSqr, int sqrStep, IppiSize roiSize,
    Ipp32s val, Ipp32s valSqr);

IppStatus ippiTiltedSqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32s* pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize,
    Ipp32s val, Ipp64f valSqr);

IppStatus ippiTiltedSqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32f* pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize,
    Ipp32f val, Ipp64f valSqr);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination integral image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSqr</i>	Pointer to the ROI of the destination integral image of pixel squares.
<i>sqrStep</i>	Distance in bytes between starts of consecutive lines in the destination integral image of pixel squares.
<i>roiSize</i>	Size of source image ROI in pixels.
<i>val</i>	The value to add to <i>pDst</i> image pixels.
<i>valSqr</i>	The value to add to <i>pSqr</i> image pixels

### Description

The function `ippiTiltedSqrIntegral` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function builds two destination image: tilted integral image *pDst* and tilted integral image of pixel squares *pSqr*.

Pixel values of  $pDst$  are computed using pixel values of the source image  $pSrc$  and the specified value  $val$  in accordance with the following formula:

$$pDst[i, j] = val + \sum_{\substack{k+1 \leq i+j-2 \\ k-1 \leq i-j+1}} pSrc[k, l]$$

Pixel values of  $pSqr$  are computed using pixel values of the source image  $pSrc$  and the specified value  $valSqr$  in accordance with the following formula:

$$pSqr[i, j] = valSqr + \sum_{\substack{k+1 \leq i+j-2 \\ k-1 \leq i-j+1}} pSrc[k, l]^2$$

where  $i, j$  are coordinates of the destination image pixels (see [Figure 11-1](#)) varying in the range  $i = 2, \dots, roiSize.height$ ,  $j = 0, \dots, roiSize.width$ . Pixel values of zero and first rows ( $i=0, 1$ ) are set to  $val$  for  $pDst$ , and to  $valSqr$  for  $pSqr$ . The size of both destination images is  $(roiSize.width + 2) \times (roiSize.height + 2)$ .

[Figure 11-1](#) shows what pixels (red circles) of the source image are used in computation new pixel values in the  $i, j$  coordinates.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>dstStep</i> or <i>sqrStep</i> is less than <i>(roiSize.width+2)</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if <i>dstStep</i> is not divisible by 4, or <i>sqrStep</i> is not divisible by 8.

## Mean

*Computes the mean of image pixel values.*

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pMean);
```

Supported values for *mod*:

8u\_C1R          16s\_C1R

#### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiMean_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pMean, IppHintAlgorithm hint);
```

#### Case 3: Masked operation on one-channel data

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean);
```

Supported values for *mod*:

8u\_C1MR          8s\_C1MR          16u\_C1MR          32f\_C1MR

#### Case 4: Operation on multi-channel integer data

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f mean[3]);
```

Supported values for *mod* :

8u\_C3R          16s\_C3R

8u\_AC4R          16s\_AC4R

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f mean[4]);
```

Supported values for *mod* :

8u\_C4R          16s\_C4R

#### Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiMean_<mod>(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f mean[3], IppHintAlgorithm hint);
```

Supported values for *mod* :

32f\_C3R  
32f\_AC4R

```
IppStatus ippiMean_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f mean[4], IppHintAlgorithm hint);
```

### Case 6: Masked operation on multi-channel data

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pMean);
```

Supported values for *mod* :

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pMean</i>	Pointer to the computed mean of pixel values.
<i>mean</i>	Array containing computed mean values for each channel of a multi-channel image.
<i>hint</i>	Option to select the algorithmic implementation of the function.

### Description

The flavors of the function `ippiMean` that perform masked operations are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. This function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the mean (average) of pixel values *pMean* for the source image *pSrc*. Computation algorithm is specified by the *hint* argument (see [Table 11-3](#)). For non-masked operations on a multi-channel image (**Case 4, 5**), the mean is computed over each channel and stored in the array *mean*.

In the mask multi-channel mode (**Case 6**), the mean is computed for a single channel of interest specified by *coi*.

The following code example shows how to use the `ippiMean` function:

### Example 11-2 Computing the Mean of Pixel Values

---

```
IppStatus mean( void ) {
    Ipp64f mean;
    Ipp8u x[5*4];
    IppiSize roi = {5,4};
    ippiSet_8u_C1R( 3, x, 5, roi );
    return ippiMean_8u_C1R( x, 5, roi, &mean );
}
```

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in the following cases: if <i>srcStep</i> has a zero or negative value; in mask mode, if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width</i> * <pixelSize>.
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

---

## Mean\_StdDev

*Computes the mean and standard deviation  
of image pixel values.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for *mod*:

8u\_C1R      8s\_C1R      16u\_C1R      32f\_C1R

#### Case 2: Masked operation on one-channel data

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean,  
    Ipp64f* pStdDev);
```

Supported values for *mod*:

8u\_C1MR      8s\_C1MR      16u\_C1MR      32f\_C1MR

#### Case 3: Operation on multi-channel data

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, int coi, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for *mod*:

8u\_C3CR      8s\_C3CR      16u\_C3CR      32f\_C3CR

#### Case 4: Masked operation on multi-channel data

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,  
    Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for *mod*:

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR



## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pMean</i>	Pointer to the computed mean of pixel values.
<i>pStdDev</i>	Pointer to the computed standard deviation of pixel values in the image.

## Description

The function `ippiMean_StdDev` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the mean and standard deviation of pixel values in the ROI of the source image *pSrc*. In the mask mode, the computation is done over pixels selected by nonzero mask values.

In the multi-channel mode, the mean is computed for a single channel of interest specified by *coi*. If any of the parameters *pMean* or *pStdDev* is not required, the zero pointer is to be passed to the corresponding parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSrc</i> or <i>pMask</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width</i> * <pixelSize>.
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

---

## RectStdDev

*Computes the standard deviation of the integral images.*

---

### Syntax

```
IppStatus ippiRectStdDev_32f_C1R(const Ipp32f* pSrc, int srcStep, const
    Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize
    roiSize, IppiRect rect);

IppStatus ippiRectStdDev_32s_C1RSfs(const Ipp32s* pSrc, int srcStep,
    const Ipp32s* pSqr, int sqrStep, Ipp32s* pDst, int dstStep, IppiSize
    roiSize, IppiRect rect, int scaleFactor);

IppStatus ippiRectStdDev_32s32f_C1R(const Ipp32s* pSrc, int srcStep,
    const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize
    roiSize, IppiRect rect);
```

### Parameters

<i>pSrc</i>	Pointer to the ROI in the source integral image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source integral image.
<i>pSqr</i>	Pointer to the ROI in the source integral image of pixel squares.
<i>sqrStep</i>	Distance in bytes between starts of consecutive lines in the source integral image of pixel squares.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of destination image ROI in pixels.
<i>rect</i>	Rectangular window.
<i>scaleFactor</i>	Factor for integer scaling.

## Description

The function `ippiRectStdDev` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the standard deviation for each pixel in the rectangular window `rect` using the integral image `pSrc` and integral image of pixel squares `pSqr`. The computations are performed in accordance with the following formulas:

$$pDst[i, j] = \sqrt{\max\left(0, \frac{sumSqr \cdot numPix - sum^2}{numPix^2}\right)}$$

where  $i, j$  are coordinates of the destination image pixels varying in the range  $i = 0, \dots, roiSize.height - 1$ ,  $j = 0, \dots, roiSize.width - 1$ ;

```
sum = pSrc[i + rect.y + rect.height, j + rect.x + rect.width] -
      - pSrc[i + rect.y, j + rect.x + rect.width] -
      - pSrc[i + rect.y + rect.height, j + rect.x] + pSrc[i + rect.y, j + rect.x];
```

```
sumSqr = pSqr[i + rect.y + rect.height, j + rect.x + rect.width] -
          - pSqr[i + rect.y, j + rect.x + rect.width] -
          - pSqr[i + rect.y + rect.height, j + rect.x] + pSqr[i + rect.y, j + rect.x];
```

```
numPix = rect.height * rect.width.
```

The minimum size of each source images `pSrc` and `pSqr` should be  $(roiSize.width + rect.x + rect.width) \times (roiSize.height + rect.y + rect.height)$ .

The source images `pSrc` and `pSqr` can be obtained by using the functions [ippiIntegral](#) or [ippiSqrIntegral](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>rect.width</code> or <code>rect.height</code> is less than or equal to zero, or if <code>rect.x</code> or <code>rect.y</code> is less than zero.

<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>sqrStep</code> is less than <code>(roiSize.width+rect.x+rect.width+1) * &lt;pixelSize&gt;</code> , or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if <code>sqrStep</code> is not divisible by 8, or one of <code>pSrc</code> and <code>dstStep</code> is not divisible by 4.

---

## TiltedRectStdDev

*Computes the standard deviation of the tilted integral images.*

---

### Syntax

```
IppStatus ippiTiltedRectStdDev_32f_C1R(const Ipp32f* pSrc, int srcStep,
    const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize
    roiSize, IppiRect rect);
```

```
IppStatus ippiTiltedRectStdDev_32s_C1RSfs(const Ipp32s* pSrc, int
    srcStep, const Ipp32s* pSqr, int sqrStep, Ipp32s* pDst, int dstStep,
    IppiSize roiSize, IppiRect rect, int scaleFactor);
```

```
IppStatus ippiTiltedRectStdDev_32s32f_C1R(const Ipp32s* pSrc, int
    srcStep, const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep,
    IppiSize roiSize, IppiRect rect);
```

### Parameters

<code>pSrc</code>	Pointer to the ROI in the source integral image.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source integral image.
<code>pSqr</code>	Pointer to the ROI in the source integral image of pixel squares.
<code>sqrStep</code>	Distance in bytes between starts of consecutive lines in the source integral image of pixel squares.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of destination image ROI in pixels.

*rect* Rectangular window.

*scaleFactor* Factor for integer scaling.

## Description

The function `ippiTiltedRectStdDev` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the standard deviation for each pixel in the rectangular window *rect* using the tilted integral image *pSrc* and tilted integral image of pixel squares *pSqr*. The computations are performed in accordance with the following formulas:

$$pDst[i, j] = \sqrt{\max\left(0, \frac{sumSqr \cdot numPix - sum^2}{numPix^2}\right)}$$

where *i, j* are coordinates of the destination image pixels varying in the range  $i = 0, \dots, roiSize.height - 1$ ,  $j = 0, \dots, roiSize.width - 1$ ;

```
sum = pSrc[i + rect.x - rect.y + rect.height + rect.width,
        j + rect.x + rect.y - rect.height + rect.width] -
      - pSrc[i + rect.x - rect.y + rect.width, j + rect.x + rect.y + rect.width] -
      - pSrc[i + rect.x - rect.y + rect.height, j + rect.x - rect.y - rect.height]
      + pSrc[i + rect.x - rect.y, j + rect.x + rect.y];
```

```
sumSqr = pSqr[i + rect.x - rect.y + rect.height + rect.width,
              j + rect.x + rect.y - rect.height + rect.width] -
        - pSqr[i + rect.x - rect.y + rect.width, j + rect.x + rect.y + rect.width] -
        - pSqr[i + rect.x - rect.y + rect.height, j + rect.x - rect.y - rect.height]
        + pSqr[i + rect.x - rect.y, j + rect.x + rect.y];
```

```
numPix = 2 * rect.height * rect.width.
```

The minimum size of each source images *pSrc* and *pSqr* should be  $(roiSize.width + rect.height + rect.width - 2) \times (roiSize.height + rect.x + rect.y + rect.height + rect.width - 2)$ .

The source images *pSrc* and *pSqr* can be obtained by using the functions [ippiTiltedIntegral](#) or [ippiTiltedSqrIntegral](#).

## Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error if one of the specified pointers is NULL.

<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>rect.width</code> or <code>rect.height</code> is less than or equal to zero, or if <code>rect.x</code> or <code>rect.y</code> is less than zero.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>sqrStep</code> is less than $(roiSize.width + rect.x + rect.width + 1) * \langle pixelSize \rangle$ , or <code>dstStep</code> is less than $roiSize.width * \langle pixelSize \rangle$ .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if <code>sqrStep</code> is not divisible by 8, or one of <code>pSrc</code> and <code>dstStep</code> is not divisible by 4.

---

## HistogramRange

*Computes the intensity histogram of an image.*

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiHistogramRange_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist, const Ipp32s* pLevels,
    int nLevels);
```

Supported values for `mod` :

```
8u_C1R      16s_C1R
```

#### Case 2: Operation on multi-channel integer data

```
IppStatus ippiHistogramRange_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist[3], const Ipp32s* pLevels[3],
    int nLevels[3]);
```

Supported values for `mod` :

```
8u_C3R      16s_C3R
8u_AC4R     16s_AC4R
```

```
IppStatus ippiHistogramRange_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist[4], const Ipp32s* pLevels[4],
    int nLevels[4]);
```

Supported values for *mod* :

```
8u_C4R      16s_C4R
```

### Case 3: Operation on one-channel floating-point data

```
IppStatus ippiHistogramRange_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist, const Ipp32f* pLevels, int nLevels);
```

### Case 4: Operation on multi-channel floating-point data

```
IppStatus ippiHistogramRange_<mod>(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[3], const Ipp32f* pLevels[3],
    int nLevels[3]);
```

Supported values for *mod* :

```
32f_C3R
32f_AC4R
```

```
IppStatus ippiHistogramRange_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp32s* pHist[4], const Ipp32f* pLevels[4],
    int nLevels[4]);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pHist</i>	Pointer to the computed histogram. In case of multi-channel data, <i>pHist</i> is an array of pointers to the histogram for each channel.
<i>pLevels</i>	Pointer to the array of level values. In case of multi-channel data, <i>pLevels</i> is an array of pointers to the level values array for each channel.
<i>nLevels</i>	Number of levels, separate for each channel.

## Description

The function `ippiHistogramRange` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the intensity histogram of an image in the ranges specified by the array (or arrays, in case of multi-channel data) *pLevels*. The histograms

computed separately for each channel are stored in the arrays *pHist*. Before calling this function, the array *pLevels* should be initialized to determine bounds of the bins for histogram computing.

Length of the arrays *pLevels* and *pHist* is defined by the *nLevels* parameter. Since *nLevels* is the number of levels, the number of histogram bins is *nLevels* - 1. Similarly, the number of values in the array *pHist* is also *nLevels* - 1.

The meaning of *pHist* and *pLevels* values can be illustrated by the following example:

*pHist*[*k*] is a number of source image pixels *pSrc*(*x*, *y*) that satisfy the condition  $pLevels[k] \leq pSrc(x, y) < pLevels[k+1]$ .

The following code example shows how to use the *ippiHistogramRange* function:

---

**Example 11-3 Computing the Histogram of an Image**

---

```
// Compute histogram of an image for 4 bins in the range [28, 127]
{
    Ipp8u img[WIDTH*HEIGHT];
    IppiSize imgSize = {WIDTH, HEIGHT};
    const Ipp32s levels[5] = {28, 50, 70, 90, 128};
    Ipp32s histo[5];

    ippiHistogramRange_8u_C1R(img, WIDTH, imgSize, levels, histo, 5);

    // after executing of the function the array histo
    // will contain a histogram in specified range.
    // Actually, four result values will be stored

    // compute cumulative histogram
    for (i = 1; i < 4; i++)
        histo[i] += histo[i-1];
}
```

---

**Return Values**

*ippStsNoErr*

Indicates no error. Any other value indicates an error or a warning.



<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error when there is not enough memory for the inner histogram.
<code>ippStsHistoNofLevelsErr</code>	Indicates an error when <i>nLevels</i> is less than 2.

---

## HistogramEven

*Computes the intensity histogram of an image using equal bins.*

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippHistogramEven_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist, Ipp32s* pLevels,
    int nLevels, Ipp32s lowerLevel, Ipp32s upperLevel);
```

Supported values for *mod* :

8u\_C1R          16s\_C1R

#### Case 2: Operation on multi-channel integer data

```
IppStatus ippHistogramEven_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist[3], Ipp32s* pLevels[3],
    int nLevels[3], Ipp32s lowerLevel[3], Ipp32s upperLevel[3]);
```

Supported values for *mod* :

8u\_C3R          16s\_C3R  
8u\_AC4R          16s\_AC4R

```
IppStatus ippHistogramEven_<mod>(const Ipp<datatype>* pSrc,
    int srcStep, IppiSize roiSize, Ipp32s* pHist[4], Ipp32s* pLevels[4],
    int nLevels[4], Ipp32s lowerLevel[4], Ipp32s upperLevel[4]);
```

Supported values for *mod* :

8u\_C4R          16s\_C4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels
<i>pHist</i>	Pointer to the computed histogram. In case of multi-channel data, <i>pHist</i> is an array of pointers to the histogram for each channel.
<i>pLevels</i>	Pointer to the array of level values. In case of multi-channel data, <i>pLevels</i> is an array of pointers to the level values array for each channel.
<i>nLevels</i>	Number of levels, separate for each channel.
<i>lowerLevel</i>	Lower level boundary, separate for each channel.
<i>upperLevel</i>	Upper level boundary, separate for each channel.

## Description

The function `ippiHistogramEven` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the intensity histogram of an image in the ranges specified by the values *lowerLevel* (inclusive), *upperLevel* (exclusive), and *nLevels*. The function operates on the assumption that all histogram bins have the same width and equal boundary values of the bins (levels).

The function stores computed histograms in the array *pHist* separately for each channel. The calculated levels are stored in the array *pLevels*.

Length of the arrays *pLevels* and *pHist* is defined by the *nLevels* parameter. Since *nLevels* is the number of levels, the number of histogram bins is *nLevels* - 1. Similarly, the number of values in the array *pHist* is also *nLevels* - 1.

The meaning of *pHist* and *pLevels* values can be illustrated by the following example:

*pHist*[*k*] is a number of source image pixels *pSrc*(*x*, *y*) that satisfy the condition

$$pLevels[k] \leq pSrc(x, y) < pLevels[k+1].$$

[Example 11-4](#) shows how to use the `ippiHistogramEven` function.

**Example 11-4 Computing the Even Histogram of an Image**

---

```
// Compute histogram of an image for 4 bins in the range [28, 127];
// compute level values for the bins.
{
    Ipp8u img[WIDTH*HEIGHT];
    IppiSize imgSize = {WIDTH, HEIGHT};
    Ipp32s levels[5], histo[5];

    ippiHistogramEven_8u_C1R(img, WIDTH, imgSize, levels,
                             histo, 5, 28, 128);

    // When the function completes operation the array histo will
    // contain a histogram in specified range, and the array
    // levels will contain the level values {28, 53, 78, 103, 128}
}
```

---

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error when there is not enough memory for the inner histogram.
<code>ippStsHistoNofLevelsErr</code>	Indicates an error when <code>nLevels</code> is less than 2.

---

## CountInRange

*Computes the number of pixels  
within the given intensity range.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiCountInRange_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, int* counts, Ipp<datatype> lowerBound,  
    Ipp<datatype> upperBound);
```

Supported values for *mod* :

8u\_C1R            32f\_C1R

#### Case 2: Operation on multi-channel data

```
IppStatus ippiCountInRange_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, int counts[3], Ipp<datatype> lowerBound[3],  
    Ipp<datatype> upperBound[3]);
```

Supported values for *mod* :

8u\_C3R            32f\_C3R  
8u\_AC4R           32f\_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>counts</i>	The computed number of pixels within the given intensity range. An array of 3 values in case of multi-channel data.
<i>lowerBound</i>	Lower limit of the intensity range.
<i>upperBound</i>	Upper limit of the intensity range.

## Description

The function `ippiCountInRange` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the number of pixels in the image which have intensity values in the range between `lowerBound` and `upperBound` (inclusive).

In case of a multi-channel image, pixels are counted within intensity range for each color channel separately, and the array `counts` of three resulting values is returned. The alpha channel values, if present, are not processed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.
<code>ippStsRangeErr</code>	Indicates an error condition if <code>lowerBound</code> exceeds <code>upperBound</code> .

---

## Min

*Computes the minimum of image pixel values.*

---

## Syntax

### Case 1: Operation on one-channel data

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype>* pMin);
```

Supported values for `mod` :

`8u_C1R`      `16s_C1R`      `32f_C1R`

### Case 2: Operation on multi-channel data

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype> min[3]);
```

Supported values for `mod` :

`8u_C3R`      `16s_C3R`      `32f_C3R`

```

8u_AC4R      16s_AC4R      32f_AC4R
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep,
IppiSize roiSize, Ipp<datatype> min[4]);

```

Supported values for *mod* :

```

8u_C4R      16s_C4R      32f_C4R

```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pMin</i>	Pointer to the minimum pixel value (for one-channel data).
<i>min</i>	Array containing minimum channel values of pixels in the source buffer (for multi-channel data).

## Description

The function `ippiMin` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the minimum pixel value *pMin* for the source image *pSrc*. In case of a multi-channel image, the minimum is computed over each channel and stored in the array *min*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pMin</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

---

## MinIndx

*Computes the minimum of image pixel values and retrieves the x and y coordinates of pixels with minimal intensity values.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp<datatype>* pMin, int* pIndexX, int* pIndexY);
```

Supported values for *mod* :

8u\_C1R          16s\_C1R          32f\_C1R

#### Case 2: Operation on multi-channel data

```
IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp<datatype> min[3], int indexX[3], int indexY[3]);
```

Supported values for *mod* :

8u\_C3R          16s\_C3R          32f\_C3R  
8u\_AC4R          16s\_AC4R          32f\_AC4R

```
IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp<datatype> min[4], int indexX[4], int indexY[4]);
```

Supported values for *mod* :

8u\_C4R          16s\_C4R          32f\_C4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pMin</i>	Pointer to the minimum pixel value (for one-channel data).
<i>min</i>	Array containing minimum color channel values of pixels in the source buffer (for multi-channel data).
<i>pIndexX</i> , <i>pIndexY</i>	Pointers to the x and y coordinates of the pixel with minimum value.

*indexX*, *indexY*      Arrays containing the x and y coordinates of pixels with minimum channel values.

## Description

The function `ippiMinIndx` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the minimum pixel value *pMin* for the source image *pSrc*. In case of a multi-channel image, the minimum is computed over each channel and stored in the array *min*. The function also retrieves the *x* and *y* coordinates of pixels on which the minimum is reached. If several pixels have equal minimum value, the coordinates of the first pixel from the start of the source buffer is returned. For multi-channel data, *indexX[k]* and *indexY[k]* are the *x* and *y* coordinates of the pixel that has the minimal intensity value of the *k*-th channel, *k* = 1,2,3,4.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of <i>pSrc</i> , <i>pMin</i> , <i>pIndexX</i> , or <i>pIndexY</i> pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

---

## Max

*Computes the maximum of image pixel values.*

---

## Syntax

### Case 1: Operation on one-channel data

```
ippStatus ippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype>* pMax);
```

Supported values for *mod* :

8u\_C1R      16s\_C1R      32f\_C1R



**Case 2: Operation on multi-channel data**

```
IppStatus ippiParam_max_mod(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype> max[3]);
```

Supported values for *mod* :

```
8u_C3R      16s_C3R      32f_C3R
8u_AC4R      16s_AC4R      32f_AC4R
```

```
IppStatus ippiParam_max_mod4(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype> max[4]);
```

Supported values for *mod* :

```
8u_C4R      16s_C4R      32f_C4
```

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels
<i>pMax</i>	Pointer to the maximum pixel value (for one-channel data).
<i>max</i>	Array containing maximum channel values of pixels in the source buffer (for multi-channel data).

**Description**

The function `ippiParam_max` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the maximum pixel value *pMax* for the source image *pSrc*. In case of a multi-channel image, the maximum is computed over each channel and stored in the array *max*.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pMax</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

---

## MaxIndx

*Computes the maximum of image pixel values and retrieves the x and y coordinates of pixels with maximal intensity values.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp<datatype>* pMax, int* pIndexX, int* pIndexY);
```

Supported values for *mod* :

8u\_C1R          16s\_C1R          32f\_C1R

#### Case 2: Operation on multi-channel data

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp<datatype> max[3], int indexX[3], int indexY[3]);
```

Supported values for *mod* :

8u\_C3R          16s\_C3R          32f\_C3R  
8u\_AC4R          16s\_AC4R          32f\_AC4R

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp<datatype> max[4], int indexX[4], int indexY[4]);
```

Supported values for *mod* :

8u\_C4R          16s\_C4R          32f\_C4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pMax</i>	Pointer to the maximum pixel value (for one-channel data).
<i>max</i>	Array containing maximum channel values of pixels in the source buffer (for multi-channel data).
<i>pIndexX</i> , <i>pIndexY</i>	Pointers to the x and y coordinates of the pixel with maximum value.

*indexX*, *indexY*      Arrays containing the x and y coordinates of pixels with maximum channel values.

## Description

The function `ippiMaxIdx` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the maximum pixel value *pMax* for the source image *pSrc*. In case of a multi-channel image, the maximum is computed over each channel and stored in the array *max*. The function also retrieves the *x* and *y* coordinates of pixels on which the maximum is reached. If several pixels have equal maximum value, the coordinates of the first pixel from the start of the source buffer is returned. For multi-channel data, *indexX[k]* and *indexY[k]* are the *x* and *y* coordinates of the pixel that has the maximal intensity value of the *k*-th channel, *k* = 1,2,3,4.

## Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if any of <i>pSrc</i> , <i>pMax</i> , <i>pIndexX</i> , or <i>pIndexY</i> pointers is NULL.
<code>ippiStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippiStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

---

## MinMax

*Computes the minimum and maximum of image pixel values.*

---

## Syntax

### Case 1: Operation on one-channel data

```
ippiStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp<datatype>* pMin, Ipp<datatype>* pMax);
```

Supported values for *mod* :

8u\_C1R      16s\_C1R      32f\_C1R

**Case 2: Operation on multi-channel data**

```
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp<datatype> min[3], Ipp<datatype> max[3]);
```

Supported values for *mod* :

8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp<datatype> min[4], Ipp<datatype> max[4]);
```

Supported values for *mod* :

8u_C4R	16s_C4R	32f_C4R
--------	---------	---------

**Parameters**

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pMin</i> , <i>pMax</i>	Pointers to the minimum and maximum pixel values (for one-channel data).
<i>min</i> , <i>max</i>	Arrays containing minimum and maximum channel values of pixels in the source buffer (for multi-channel data).

**Description**

The function `ippiMinMax` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the minimum and maximum pixel values *pMin* and *pMax* for the source image *pSrc*. In case of a multi-channel image, the minimum and maximum is computed over each channel and stored in the arrays *min* and *max*.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> , <i>pMin</i> , or <i>pMax</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

## MinMaxIndx

*Calculates minimum and maximum pixel values and their indexes in selected image rectangle.*

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal,
    IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for *mod* :

8u\_C1R      8s\_C1R      16u\_C1R      32f\_C1R

#### Case 2: Masked operation on one-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f* pMinVal,
    Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for *mod* :

8u\_C1MR      8s\_C1MR      16u\_C1MR      32f\_C1MR

#### Case 3: Operation on multi-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal,
    IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for *mod* :

8u\_C3CR      8s\_C3CR      16u\_C3CR      32f\_C3CR

#### Case 4: Masked operation on multi-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex,
    IppiPoint* pMaxIndex);
```

Supported values for *mod* :

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pMinVal</i>	Pointer to the variable that returns the value of the minimum pixel.
<i>pMaxVal</i>	Pointer to the variable that returns the value of the maximum pixel.
<i>pMinIndex</i>	Pointer to the variable that returns the index of the minimum value found.
<i>pMaxIndex</i>	Pointer to the variable that returns the index of the maximum value found.

## Description

The function `ippiMinMaxIdx` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function finds minimum and maximum pixel values and their indexes in an image ROI or in an arbitrary image region defined by nonzero mask values. If there are several minima and maxima in the selected area, the function returns the top leftmost positions.

If the specified region in the mask mode is empty, that is, the mask image is filled with zeros, then the function returns  $\{minIndex, maxIndex\} = \{0, 0\}$ ,  $minVal = maxVal = 0$ .

If any of the parameters *pMinVal*, *pMaxVal*, *pMinIndex*, or *pMaxIndex* is not required, the zero pointer is to be passed to the corresponding parameter.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pMask</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> or <i>maskStep</i> is less than $roiSize.width * <pixelSize>$ .

<code>ippStsNotEvenStepErr</code>	Indicates an error when steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <code>coi</code> is not 1, 2, or 3.

## Image Moments

Spatial and central moments are important statistical properties of an image. The spatial moment  $M_U(m,n)$  of order  $(m,n)$  is defined as follows:

$$M_U(m, n) = \sum_j \sum_k x_k^m y_j^n P_{j, k}$$

where the summation is performed for all rows and columns in the image;  $P_{j,k}$  are pixel values;  $x_k$  and  $y_j$  are pixel coordinates;  $m$  and  $n$  are integer power exponents that define the moment order.

The central moment  $U_U(m,n)$  is the spatial moment computed relative to the “center of gravity”  $(x_0, y_0)$ :

$$U_U(m, n) = \sum_j \sum_k (x_k - x_0)^m (y_j - y_0)^n P_{j, k}$$

where  $x_0 = M_U(1,0)/M_U(0,0)$  and  $y_0 = M_U(0,1)/M_U(0,0)$ .

The normalized spatial moment  $M(m,n)$  and central moment  $U(m,n)$  are defined as follows:

$$M(m, n) = \frac{M_U(m, n)}{M_U(0, 0)^{\frac{m+n+2}{2}}}$$

$$U(m, n) = \frac{U_U(m, n)}{U_U(0, 0)^{\frac{m+n+2}{2}}}$$

The Intel IPP functions support moments of order  $(m, n)$  with  $0 \leq m + n \leq 3$ . The computation of seven invariant Hu moments derived from the second and third order moments is also supported. All computed moments are stored in context structures of type `IppiMomentState_64s` (for integer versions) or `IppiMomentState_64f` (for floating point versions).

Most Intel IPP functions for computing image moments have code branches that implement different algorithms to compute the results. You can choose the desired code variety to be used by the given function by setting the *hint* argument to one of the following values that are listed in [Table 11-3](#):

**Table 11-3 Hint Arguments for Image Moment Functions**

Value	Description
<code>ippAlgHintNone</code>	The computation algorithm will be chosen by the internal function logic.
<code>ippAlgHintFast</code>	Fast algorithm must be used. The output results will be less accurate.
<code>ippAlgHintAccurate</code>	High accuracy algorithm must be used. The function will need more time to execute.

---

## MomentInitAlloc

*Allocates memory and initializes the moment context structure.*

---

### Syntax

```
IppStatus ippimomentInitAlloc_64f(IppiMomentState_64f** pState,
    IppHintAlgorithm hint);
IppStatus ippimomentInitAlloc_64s(IppiMomentState_64s** pState,
    IppHintAlgorithm hint);
```

### Parameters

*pState*                      Pointer to the structure for storing moment values.

*hint*                        Option to select the algorithmic implementation of the function.



## Description

The function `ippiMomentInitAlloc` is declared in the `ippi.h` file. This function initializes the structure that is needed for the function `ippiMoments` to store the computed image moments. Different functions are used to allocate structures for integer and floating-point data. Computation algorithm is specified by *hint* argument (see [Table 11-3](#)).

If the initialization is successful, the `ippiMomentInitAlloc` function sets the *pState* pointer to the allocated structure of `IppiMomentState` type.

If memory allocation failed, the function returns `ippStsMemAllocErr` error status code.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsMemAllocErr</code>	Indicates an error condition in case of memory allocation failure.

---

## MomentFree

*Frees memory allocated by the function*

`ippiMomentInitAlloc`.

---

## Syntax

```
IppStatus ippiMomentFree_64f(IppiMomentState_64f* pState);  
IppStatus ippiMomentFree_64s(IppiMomentState_64s* pState);
```

## Parameters

<i>pState</i>	Pointer to the structure for image moments storage.
---------------	---

## Description

The function `ippiMomentFree` is declared in the `ippi.h` file. Use this function to deallocate the previously initialized *pState* structure that stored the image moments data. Different functions are used to deallocate structures for integer and floating-point data.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.

---

## MomentGetStateSize

*Computes the size of the external buffer for the moment context structure.*

---

### Syntax

```
IppStatus ippimomentGetStateSize_64s(IppHintAlgorithm hint, int* pSize);
```

### Parameters

<i>pSize</i>	Pointer to the computed value of the buffer size.
<i>hint</i>	Option to select the algorithmic implementation of the function.

### Description

The function `ippimomentGetStateSize` is declared in the `ippi.h` file. Use this function to determine the size of the external work buffer for the moment context structure to be initialized by the function [ippiMomentInit](#). Computation algorithm is specified by *hint* argument (see [Table 11-3](#) ).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSize</i> pointer is NULL.

---

## MomentInit

*Initializes the moment context structure.*

---

### Syntax

```
IppStatus ippMomentInit_64s(IppiMomentState_64s* pState,  
    IppHintAlgorithm hint);
```

### Parameters

<i>pState</i>	Pointer to the structure for storing moment values.
<i>hint</i>	Option to select the algorithmic implementation of the function.

### Description

The function `ippMomentInit` is declared in the `ippi.h` file. This function initializes the structure that is needed for the function `ippMoments` to store the computed image moments. Computation algorithm is specified by *hint* argument (see [Table 11-3](#) ).

The structure is allocated in the external buffer. The size of this buffer can be computed by the function [ippMomentGetStateSize](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> pointer is NULL.

---

## Moments

*Computes all image moments of order 0 to 3  
and Hu moment invariants.*

---

### Syntax

#### Case 1: Computation of floating-point results

```
IppStatus ippMoments64f_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, IppiMomentState_64f* pState);
```

Supported values for *mod* :

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_AC4R	32f_AC4R

### Case 2: Computation of integer results

```
IppStatus ippIMoments64s_<mod>(const Ipp8u* pSrc, int srcStep,  
    IppiSize roiSize, IppiMomentState_64s* pState);
```

Supported values for *mod* :

8u_C1R
8u_C3R
8u_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pState</i>	Pointer to the structure that stores image moments.

### Description

The function `ippIMoments` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes all spatial and central moments of order 0 to 3 for the source image *pSrc*. The seven Hu moment invariants are also computed.

Different functions, `ippIMoments64s` and `ippIMoments64f`, are used to compute image moments in integer and floating-point formats, respectively.

The `ippIMoments` function computes spatial moment values relative to the image point referred to by *pSrc*. Note that this point is the ROI origin and may not coincide with the entire image origin. If you need to obtain spatial moment values relative to the actual image origin, use [ippiGetSpatialMoment](#) functions to recalculate them.

The moments' values are stored in the *pState* structure. To retrieve a particular moment value, use one of the functions described in the sections that follow.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pState</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.

---

## GetSpatialMoment

*Retrieves image spatial moment of the specified order, computed by `ippiMoments`.*

---

### Syntax

```
IppStatus ippGetSpatialMoment_64f(const IppiMomentState_64f* pState,
    int mOrd, int nOrd, int nChannel, IppiPoint roiOffset,
    Ipp64f* pValue);
IppStatus ippGetSpatialMoment_64s(const IppiMomentState_64s* pState,
    int mOrd, int nOrd, int nChannel, IppiPoint roiOffset,
    Ipp64s* pValue, int scaleFactor);
```

### Parameters

<i>pState</i>	Pointer to the structure that stores image moments.
<i>mOrd</i> , <i>nOrd</i>	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$ .
<i>nChannel</i>	The channel for which the moment is returned.
<i>roiOffset</i>	Offset in pixels of the ROI origin (top left corner) from the image origin.
<i>pValue</i>	Pointer to the retrieved moment value.

*scaleFactor*                      Factor to scale the moment value in case of integer data.

### Description

The function `ippiGetSpatialMoment` is declared in the `ippi.h` file. This function returns the pointer *pValue* to the spatial moment that was previously computed by the [ippiMoments](#) function. All spatial moment values are computed by `ippiMoments` relative to the image ROI origin.

You may also obtain spatial moment values relative to different point in the image, using the appropriate *roiOffset* settings.

The moment order is specified by the integer exponents *mOrd*, *nOrd*.

Different functions are used to retrieve spatial moment in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified *scaleFactor*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> or <i>pValue</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>mOrd</i> + <i>nOrd</i> is greater than 3, or <i>nChannel</i> has an illegal value.

---

## GetNormalizedSpatialMoment

*Retrieves the normalized value of the image spatial moment computed by `ippiMoments`.*

---

### Syntax

```
IppStatus ippiGetNormalizedSpatialMoment_64f(IppiMomentState_64f*
    pState, int mOrd, int nOrd, int nChannel, IppiPoint roiOffset,
    Ipp64f* pValue);
```

```
IppStatus ippiGetNormalizedSpatialMoment_64s(IppiMomentState_64s*  
    pState, int mOrd, int nOrd, int nChannel, IppiPoint roiOffset,  
    Ipp64s* pValue, int scaleFactor);
```

## Parameters

<i>pState</i>	The structure that stores image moments.
<i>mOrd</i> , <i>nOrd</i>	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$ .
<i>nChannel</i>	The channel for which the moment is returned.
<i>roiOffset</i>	Offset in pixels of the ROI origin (top left corner) from the image origin.
<i>pValue</i>	Pointer to the returned normalized moment value.
<i>scaleFactor</i>	Factor to scale the moment value in case of integer data.

## Description

The function `ippiGetNormalizedSpatialMoment` is declared in the `ippi.h` file. This function normalizes the spatial moment value that was previously computed by the [ippiMoments](#) function, and returns the pointer *pValue* to the normalized moment. See [Image Moments](#) for details of moments normalization. The moment order (*mOrd*, *nOrd*) is specified by integer power exponents.

All spatial moment values are computed by `ippiMoments` relative to the image ROI origin. You may also obtain normalized spatial moment values relative to different point in the image, using the appropriate *roiOffset* settings.

Different functions are used to retrieve normalized spatial moment in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified *scaleFactor*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> or <i>pValue</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.

<code>ippStsMoment00ZeroErr</code>	Indicates an error condition if $M(0,0)$ value is close to zero.
<code>ippStsSizeErr</code>	Indicates an error condition if $mOrd + nOrd$ is greater than 3, or <code>nChannel</code> has an illegal value.

---

## GetCentralMoment

*Retrieves image central moment computed by `ippiMoments`.*

---

### Syntax

```

IppStatus ippGetCentralMoment_64f(const IppiMomentState_64f* pState,
    int mOrd, int nOrd, int nChannel, Ipp64f* pValue);
IppStatus ippGetCentralMoment_64s(const IppiMomentState_64s* pState,
    int mOrd, int nOrd, int nChannel, Ipp64s* pValue,
    int scaleFactor);

```

### Parameters

<code>pState</code>	The structure that stores image moments.
<code>mOrd, nOrd</code>	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$ .
<code>nChannel</code>	The channel for which the moment is returned.
<code>pValue</code>	Pointer to the returned moment value.
<code>scaleFactor</code>	Factor to scale the moment value in case of integer data.

### Description

The function `ippGetCentralMoment` is declared in the `ippi.h` file. This function returns the pointer `pValue` to the central moment previously computed by the [ippiMoments](#) function. The moment order is specified by the integer exponents `mOrd`, `nOrd`.

Different functions are used to retrieve central moment in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified `scaleFactor`.



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pState</i> or <i>pValue</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>mOrd</i> + <i>nOrd</i> is greater than 3, or <i>nChannel</i> has an illegal value.

---

## GetNormalizedCentralMoment

*Retrieves the normalized value of the image central moment computed by `ippiMoments`.*

---

### Syntax

```
IppStatus ippiGetNormalizedCentralMoment_64f(IppiMomentState_64f*  
    pState, int mOrd, int nOrd, int nChannel, Ipp64f* pValue);  
IppStatus ippiGetNormalizedCentralMoment_64s(IppiMomentState_64s*  
    pState, int mOrd, int nOrd, int nChannel, Ipp64s* pValue,  
    int scaleFactor);
```

### Parameters

<i>pState</i>	The structure that stores image moments.
<i>mOrd</i> , <i>nOrd</i>	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$ .
<i>nChannel</i>	The channel for which the moment is returned.
<i>pValue</i>	Pointer to the returned moment value.
<i>scaleFactor</i>	Factor to scale the moment value in case of integer data.

## Description

The function `ippiGetNormalizedCentralMoment` is declared in the `ippi.h` file. This function normalizes the central moment value that was previously computed by the [ippiMoments](#) function, and returns the pointer `pValue` to the normalized moment. The moment order (`mOrd`, `nOrd`) is specified by the integer power exponents. See [Image Moments](#) for details of moments normalization.

Different functions are used to retrieve normalized central moment in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified `scaleFactor`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pState</code> or <code>pValue</code> pointer is <code>NULL</code> .
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid structure is passed.
<code>ippStsMoment00ZeroErr</code>	Indicates an error condition if $M(0,0)$ value is close to zero.

---

## GetHuMoments

*Retrieves image Hu moment invariants computed by `ippiMoments` function.*

---

## Syntax

```
IppStatus ippiGetHuMoments_64f(IppiMomentState_64f* pState,  
    int nChannel, IppiHuMoment_64f* pHm);  
  
IppStatus ippiGetHuMoments_64s(IppiMomentState_64s* pState,  
    int nChannel, IppiHuMoment_64s* pHm, int scaleFactor);
```

## Parameters

<code>pState</code>	Pointer to the structure that stores image moments.
<code>nChannel</code>	The channel for which the moment is returned.
<code>pHm</code>	Pointer to the array containing the Hu moment invariants.

*scaleFactor*                      Factor to scale moment values in case of integer data.

## Description

The function `ippiGetHuMoments` is declared in the `ippi.h` file. This function returns the pointer *pHm* to the array of seven Hu moment invariants previously computed by the [ippiMoments](#) function. Different functions are used to retrieve Hu moments in integer and floating-point formats, respectively. In case of integer data, the result may be scaled by the specified *scaleFactor*.

The following code example shows how to compute Hu moment values for images that have horizontal and vertical lines.

### Example 11-5 Computing Hu moments

```
IppStatus mom8u( void ) {
    IppStatus stH, stV;
    Ipp8u x[64];
    IppiSize roiA={8,8}, roiH={8,1}, roiV={1,8};
    IppiHuMoment_64f hmH, hmV;
    IppiMomentState_64f* ctx;

    stH = ippiMomentInitAlloc_64f( &ctx, ippAlgHintNone );

    ippiSet_8u_C1R( 0, x, 8, roiA );
    ippiSet_8u_C1R( 3, x, 8, roiH );
    stH = ippiMoments64f_8u_C1R( x, 8, roiA, ctx );
    ippiGetHuMoments_64f( ctx, 0, hmH );

    ippiSet_8u_C1R( 0, x, 8, roiA );
    ippiSet_8u_C1R( 3, x, 8, roiV );
    stV = ippiMoments64f_8u_C1R( x, 8, roiA, ctx );
    ippiGetHuMoments_64f( ctx, 0, hmV );

    ippiMomentFree_64f( ctx );
    return ippStsNoErr==stH ? stV : stH;
}
```

## Return Values

`ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.

`ippStsNullPtrErr` Indicates an error condition if `pState` or `pHm` pointer is NULL.

`ippStsContextMatchErr` Indicates an error condition if a pointer to an invalid structure is passed.

`ippStsMoment00ZeroErr` Indicates an error condition if  $M(0,0)$  value is close to zero.

## Image Norms

The functions described in this section compute the following norms of the image pixel values:

- Infinity norm (the largest absolute pixel value).
- L1 norm (the sum of absolute pixel values)
- L2 norm (the square root of the sum of squared pixel values)

Functions of this group also help you compute the norm of differences in pixel values of two input images as well as the relative error for two input images.

---

## Norm\_Inf

*Computes the infinity norm  
of image pixel values.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for `mod` :

`8u_C1R`      `16s_C1R`      `32s_C1R`      `32f_C1R`

#### Case 2: Masked operation on one-channel data

```
IppStatus ippNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C1MR      8s\_C1MR      16u\_C1MR      32f\_C1MR

### Case 3: Operation on multi-channel data

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

8u\_C3R      16s\_C3R      32f\_C3R  
8u\_AC4R      16s\_AC4R      32f\_AC4R

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

8u\_C4R      16s\_C4R      32f\_C4R

### Case 4: Masked operation on multi-channel data

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed infinity norm of pixel values.
<i>value</i>	An array containing the computed infinity norms of channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.

## Description

The flavors of the function `ippiNorm_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)) and computes the infinity norm *pValue* (*pNorm* for the mask mode) for the source image *pSrc*. This norm is defined as the largest absolute pixel value in an image. In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (**Case 3**), the norm is computed separately for each channel and stored in the array *value*.

In the mask multi-channel mode (**Case 4**), the norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in the following cases: if <i>srcStep</i> has a zero or negative value; in mask mode, if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

---

## Norm\_L1

*Computes the L1- norm  
of image pixel values.*

---

## Syntax

### Case 1: Operation on one-channel integer data

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
                             IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

8u\_C1R            16s\_C1R

### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNorm_L1_32f_C1R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

### Case 3: Masked operation on one-channel data

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C1MR            8s\_C1MR            16u\_C1MR            32f\_C1MR

### Case 4: Operation on multi-channel integer data

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

8u\_C3R            16s\_C3R  
8u\_AC4R            16s\_AC4R

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

8u\_C4R            16s\_C4R

### Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNorm_L1_<mod>(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

Supported values for *mod* :

32f\_C3R  
32f\_AC4R

```
IppStatus ippiNorm_L1_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

### Case 6: Masked operation on multi-channel data

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed L1- norm of pixel values.
<i>value</i>	An array containing the computed L1- norms of channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The flavors of the function `ippiNorm_L1` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L1- norm *pValue* (*pNorm* in mask mode) for the source image *pSrc*. This norm is defined as the sum of absolute pixel values in an image. Computation algorithm is specified by the *hint* argument (see [Table 11-3](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (**Case 4, 5**), the norm is computed separately for each channel and stored in the array *value*.

In the mask multi-channel mode (**Case 6**), the norm is computed for a single channel of interest specified by *coi*.

The following [Example 11-6](#) demonstrates how an image norm can be computed.



**Example 11-6 Computing the Image Norm**

---

```
IppStatus norm( void ) {  
    Ipp64f sum, normL1;  
    Ipp8u x[5*4];  
    IppiSize roi = {5,4};  
    ippiSet_8u_C1R( 1, x, 5, roi );  
    ippiSum_8u_C1R( x, 5, roi, &sum);  
    return ippiNorm_L1_8u_C1R( x, 5, roi, &normL1 );  
}
```

---

**Return Values**

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in the following cases: if <i>srcStep</i> has a zero or negative value; in mask mode, if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width</i> * <pixelSize>.
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## Norm\_L2

*Computes the L2- norm of image pixel values.*

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

8u\_C1R            16s\_C1R

#### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNorm_L2_32f_C1R(const Ipp32f* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

#### Case 3: Masked operation on one-channel data

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C1MR            8s\_C1MR            16u\_C1MR            32f\_C1MR

#### Case 4: Operation on multi-channel integer data

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

8u\_C3R            16s\_C3R  
8u\_AC4R            16s\_AC4R

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

8u\_C4R            16s\_C4R

#### Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNorm_L2_<mod>(const Ipp32f* pSrc, int srcStep,  
    IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

Supported values for *mod* :

32f\_C3R

32f\_AC4R

```
IppStatus ippNorm_L2_32f_C4R(const Ipp32f* pSrc, int srcStep,
    IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

### Case 6: Masked operation on multi-channel data

```
IppStatus ippNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C3CMR

8s\_C3CMR

16u\_C3CMR

32f\_C3CMR

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed L2- norm of pixel values.
<i>value</i>	An array containing the computed L2- norms of channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The flavors of the function `ippNorm_L2` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L2- norm *pValue* (*pNorm* in mask mode) for the source image *pSrc*. This norm is defined as the square root of the sum of squared pixel values in an image. Computation algorithm is specified by the *hint* argument (see [Table 11-3](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (**Case 4,5**), the norm is computed separately for each channel and stored in the array *value*.

In the mask multi-channel mode (**Case 6**), the norm is computed for a single channel of interest specified by *coi*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in the following cases: if <i>srcStep</i> has a zero or negative value; if <i>srcStep</i> or <i>maskStep</i> is less than <i>roiSize.width</i> * <pixelSize> in mask mode.
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

---

## NormDiff\_Inf

*Computes the infinity norm of differences between pixel values of two images.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

`8u_C1R`      `16s_C1R`      `32f_C1R`

**Case 2: Masked operation on one-channel data**

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C1MR      8s\_C1MR      16u\_C1MR      32f\_C1MR

**Case 3: Operation on multi-channel data**

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

8u\_C3R      16s\_C3R      32f\_C3R  
8u\_AC4R      16s\_AC4R      32f\_AC4R

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

8u\_C4R      16s\_C4R      32f\_C4R

**Case 4: Masked operation on multi-channel data**

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR

**Parameters**

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.

<i>pValue</i>	Pointer to the computed infinity norm of difference between pixel values.
<i>value</i>	An array containing the computed infinity norms of difference between corresponding channel values in case of multi-channel data.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.

### Description

The flavors of the function `ippiNormDiff_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the infinity norm *pValue* (*pNorm* in the mask mode) of differences between pixel values of the two source images *pSrc1* and *pSrc2*. This norm is defined as the largest absolute value of differences:

$$\text{norm} = \max |pSrc1 - pSrc2|$$

In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (**Case 3**), the norm is computed separately for each pair of corresponding channels and stored in the array *value*.

In the mask multi-channel mode (**Case 4**), the norm is computed for a single channel of interest specified by *coi*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in the following cases: if <i>src1Step</i> or <i>src2Step</i> has a zero or negative value; in mask mode, if <i>src1Step</i> , <i>src2Step</i> , or <i>maskStep</i> is less than <i>roiSize.width</i> * <code>&lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## NormDiff\_L1

*Computes the L1- norm of differences  
between pixel values of two images.*

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

8u\_C1R          16s\_C1R

#### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormDiff_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue,
    IppHintAlgorithm hint);
```

#### Case 3: Masked operation on one-channel data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask,
    int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C1MR          8s\_C1MR          16u\_C1MR          32f\_C1MR

#### Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

8u\_C3R          16s\_C3R  
8u\_AC4R          16s\_AC4R

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

8u\_C4R            16s\_C4R

### Case 5: Operation on multi-channel floating-point data

```

IppStatus ippiNormDiff_L1_<mod>(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);

```

Supported values for *mod* :

32f\_C3R  
32f\_AC4R

```

IppStatus ippiNormDiff_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4],
    IppHintAlgorithm hint);

```

### Case 6: Masked operation on multi-channel data

```

IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask,
    int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);

```

Supported values for *mod* :

8u\_C3CMR        8s\_C3CMR        16u\_C3CMR        32f\_C3CMR

## Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed L1- norm of difference between pixel values.
<i>value</i>	An array containing the computed L1- norms of difference between channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.



## Description

The flavors of the function `ippiNormDiff_L1` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L1-norm *pValue* (*pNorm* in the mask mode) of differences between pixel values of the two source image buffers *pSrc1* and *pSrc2*.

This norm is defined as the sum of absolute values of differences:

$$\text{norm} = \sum |pSrc1 - pSrc2|$$

Computation algorithm is specified by the *hint* argument (see [Table 11-3](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (**Case 4,5**), the norm is computed separately for each pair of the corresponding channels and stored in the array *value*.

In the mask multi-channel mode (**Case 6**), the norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in the following cases: if <i>src1Step</i> or <i>src2Step</i> has a zero or negative value; in mask mode, if <i>src1Step</i> , <i>src2Step</i> , or <i>maskStep</i> is less than <i>roiSize.width</i> * <pixelSize>.
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## NormDiff\_L2

*Computes the L2- norm of differences  
between pixel values of two images.*

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,  
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,  
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

8u\_C1R            16s\_C1R

#### Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormDiff_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step,  
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue,  
    IppHintAlgorithm hint);
```

#### Case 3: Masked operation on one-channel data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,  
    int src1Step, const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask,  
    int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C1MR            8s\_C1MR            16u\_C1MR            32f\_C1MR

#### Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,  
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,  
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

8u\_C3R            16s\_C3R  
8u\_AC4R            16s\_AC4R

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,  
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,  
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

8u\_C4R            16s\_C4R

### Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
```

Supported values for *mod* :

32f\_C3R  
32f\_AC4R

```
IppStatus ippiNormDiff_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4],
    IppHintAlgorithm hint);
```

### Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask,
    int maskStep, IppiSize roiSize, int coi, Ipp64f* norm);
```

Supported values for *mod* :

8u\_C3CMR        8s\_C3CMR        16u\_C3CMR        32f\_C3CMR

## Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed L2- norm of difference between pixel values.
<i>value</i>	An array containing the computed L2- norms of difference between channel values in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The flavors of the function `ippiNormDiff_L2` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L2-norm *pValue* (*pNorm* in the mask mode) of differences between pixel values of the two source image buffers *pSrc1* and *pSrc2*.

This norm is defined as the square root of the sum of squared differences:

$$\text{norm} = \sqrt{\sum |pSrc1 - pSrc2|^2}.$$

Computation algorithm is specified by the *hint* argument (see [Table 11-3](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (**Case 4,5**), the norm is computed separately for each pair of the corresponding channels and stored in the array *value*.

In the mask multi-channel mode (**Case 6**), the norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in the following cases: if <i>src1Step</i> or <i>src2Step</i> has a zero or negative value; in mask mode, if <i>src1Step</i> , <i>src2Step</i> , or <i>maskStep</i> is less than <i>roiSize.width</i> * <pixelSize>.
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.

## NormRel\_Inf

*Computes the relative error for the infinity norm of differences between pixel values of two images.*

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

8u\_C1R      16s\_C1R      32f\_C1R

#### Case 2: Masked operation on one-channel data

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C1MR      8s\_C1MR      16u\_C1MR      32f\_C1MR

#### Case 3: Operation on multi-channel data

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[3]);
```

Supported values for *mod* :

8u\_C3R      16s\_C3R      32f\_C3R  
8u\_AC4R      16s\_AC4R      32f\_AC4R

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f value[4]);
```

Supported values for *mod* :

8u\_C4R      16s\_C4R      32f\_C4R

**Case 4: Masked operation on multi-channel data**

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
    Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR

**Parameters**

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pValue</i>	Pointer to the computed relative error value.
<i>value</i>	An array containing the computed relative error values for separate channels in case of multi-channel data.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pNorm</i>	Pointer to the computed relative norm value in the mask mode.

**Description**

The flavors of the function `ippiNormRel_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the infinity norm of differences between pixel values of two source buffers *pSrc1* and *pSrc2*. This norm is defined as the largest absolute pixel value in an image.

The output relative error *pValue* (*pNorm* in the mask mode) is then formed by dividing the computed norm of differences by the infinity norm of the second source image buffer *pSrc2*. In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (**Case 3**), the relative norm is computed separately for each pair of corresponding channels and stored in the array *value*.

In the mask multi-channel mode (**Case 4**), the relative norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in the following cases: if <code>src1Step</code> or <code>src2Step</code> has a zero or negative value; in mask mode, if <code>src1Step</code> , <code>src2Step</code> , or <code>maskStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <code>coi</code> is not 1, 2, or 3.
<code>ippStsDivByZero</code>	Indicates a warning when the infinity norm of <code>pSrc2</code> has a zero value.

---

## NormRel\_L1

*Computes the relative error for the L1 norm of differences between pixel values of two images.*

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize,
    Ipp64f* pValue);
```

Supported values for `mod` :

```
8u_C1R      16s_C1R
```

#### Case 2: Operation on one-channel floating-point data

```
IppStatus ippNormRel_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue,
    IppHintAlgorithm hint);
```

### Case 3: Masked operation on one-channel data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C1MR      8s\_C1MR      16u\_C1MR      32f\_C1MR

### Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize,
    Ipp64f value[3]);
```

Supported values for *mod* :

8u\_C3R      16s\_C3R  
8u\_AC4R      16s\_AC4R

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize,
    Ipp64f value[4]);
```

Supported values for *mod* :

8u\_C4R      16s\_C4R

### Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
```

Supported values for *mod* :

32f\_C3R  
32f\_AC4R

```
IppStatus ippiNormRel_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4],
    IppHintAlgorithm hint);
```

### Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* norm);
```



Supported values for *mod* :

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR

## Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed relative error value.
<i>value</i>	An array containing the computed relative error values for separate channels in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed relative norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The flavors of the function `ippiNormRel_L1` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L1- norm of differences between pixel values of two source buffers *pSrc1* and *pSrc2*. This norm is defined as the sum of absolute pixel values in an image. The output relative error *pValue* (*pNorm* in the mask mode) is then formed by dividing the computed norm of differences by the L1- norm of the second source image buffer *pSrc2*. Computation algorithm is specified by the *hint* argument (see [Table 11-3](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (**Cases 4, 5**), the relative norm is computed separately for each pair of corresponding channels and stored in the array *value*.

In the mask multi-channel mode (**Case 6**), the relative norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition in the following cases: if <i>src1Step</i> or <i>src2Step</i> has a zero or negative value; in mask mode, if <i>src1Step</i> , <i>src2Step</i> , or <i>maskStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <i>coi</i> is not 1, 2, or 3.
<code>ippStsDivByZero</code>	Indicates a warning when the L1 norm of <i>pSrc2</i> has a zero value.

---

## NormRel\_L2

*Computes the relative error for the L2 norm of differences between pixel values of two images.*

---

### Syntax

#### Case 1: Operation on one-channel integer data

```
IppStatus ippNormRel_L2_<mod>(const Ipp<datatype>* pSrc1,
    int src1Step, const Ipp<datatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp64f* pValue);
```

Supported values for *mod* :

```
8u_C1R      16s_C1R
```

#### Case 2: Operation on one-channel floating-point data

```
IppStatus ippNormRel_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue,
    IppHintAlgorithm hint);
```

#### Case 3: Masked operation on one-channel data

```
IppStatus ippNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C1MR      8s\_C1MR      16u\_C1MR      32f\_C1MR

#### Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize,
    Ipp64f value[3]);
```

Supported values for *mod* :

8u\_C3R      16s\_C3R  
8u\_AC4R      16s\_AC4R

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize,
    Ipp64f value[4]);
```

Supported values for *mod* :

8u\_C4R      16s\_C4R

#### Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormRel_L2_<mod>(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3],
    IppHintAlgorithm hint);
```

Supported values for *mod* :

32f\_C3R  
32f\_AC4R

```
IppStatus ippiNormRel_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4],
    IppHintAlgorithm hint);
```

#### Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep,
    IppiSize roiSize, int coi, Ipp64f* pNorm);
```

Supported values for *mod* :

8u\_C3CMR      8s\_C3CMR      16u\_C3CMR      32f\_C3CMR

### Parameters

*pSrc1*, *pSrc2*      Pointers to the source images ROI.

<i>src1Step</i> , <i>src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>pMask</i>	Pointer to the mask image.
<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the mask image.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>coi</i>	Channel of interest (for color images only); can be 1, 2, or 3.
<i>pValue</i>	Pointer to the computed relative error value.
<i>value</i>	An array containing the computed relative error values for separate channels in case of multi-channel data.
<i>pNorm</i>	Pointer to the computed relative norm value in the mask mode.
<i>hint</i>	Option to select the algorithmic implementation of the function.

## Description

The flavors of the function `ippiNormRel_L2` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see [Regions of Interest in Intel IPP](#)). It computes the L2- norm of differences between pixel values of two source buffers *pSrc1* and *pSrc2*. This norm is defined as the square root of the sum of squared pixel values in an image. The output relative error *pValue* (*pNorm* in the mask mode) is then formed by dividing the computed norm of differences by the L2- norm of the second source image buffer *pSrc2*. Computation algorithm is specified by the *hint* argument (see [Table 11-3](#)). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (**Cases 4, 5**), the relative norm is computed separately for each pair of corresponding channels and stored in the array *value*.

In the mask multi-channel mode (**Case 6**), the relative norm is computed for a single channel of interest specified by *coi*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

<code>ippStsStepErr</code>	Indicates an error condition in the following cases: if <code>src1Step</code> or <code>src2Step</code> has a zero or negative value; in mask mode, if <code>src1Step</code> , <code>src2Step</code> , or <code>maskStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
<code>ippStsCOIErr</code>	Indicates an error when <code>coi</code> is not 1, 2, or 3.
<code>ippStsDivByZero</code>	Indicates a warning when the L2 norm of <code>pSrc2</code> has a zero value.

## Image Quality Index

Intel IPP functions described in this section compute the universal image quality index [Wang02] that may be used as image and video quality distortion measure. It is mathematically defined by modeling the image distortion relative to the reference image as a combination of three factors: loss of correlation, luminance distortion, and contrast distortion.

If two images  $f$  and  $g$  are considered as a matrices with  $M$  column and  $N$  rows containing pixel values  $f[i,j]$ ,  $g[i,j]$ , respectively (  $0 \leq i < M$  ,  $0 \leq j < N$  ), the universal image quality index  $Q$  may be calculated as a product of three components:

$$Q = \frac{\sigma_{fg}}{\sigma_f \sigma_g} \cdot \frac{2\bar{f}\bar{g}}{(\bar{f})^2 + (\bar{g})^2} \cdot \frac{2\sigma_f \sigma_g}{\sigma_f^2 + \sigma_g^2}$$

where

$$\bar{f} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} f[i,j] \quad \bar{g} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} g[i,j]$$

$$\sigma_{fg} = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (f[i,j] - \bar{f})(g[i,j] - \bar{g})$$

$$\sigma_f^2 = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (f[i,j] - \bar{f})^2 \quad \sigma_g^2 = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (g[i,j] - \bar{g})^2$$

The first component is the correlation coefficient, which measures the degree of linear correlation between images  $f$  and  $g$ . It varies in the range  $[-1, 1]$ . The best value 1 is obtained when  $f$  and  $g$  are linearly related, which means that  $g[i,j] = af[i,j] + b$  for all possible values of  $i$  and  $j$ .

The second component, with a value range of  $[0, 1]$ , measures how close the mean luminance is between images.

Since  $\sigma_f$  and  $\sigma_g$  can be considered as estimates of the contrast of  $f$  and  $g$ , the third component measures how similar the contrasts of the images are. The value range for this component is also  $[0, 1]$ .

The range of values for the index  $Q$  is  $[-1, 1]$ . The best value 1 is achieved if and only if the images are identical.

---

## QualityIndex

*Computes the universal image quality index.*

---

### Syntax

#### Case 1: Operation on one-channel data

```
IppStatus ippiQualityIndex_<mod>(const Ipp<srcDatatype>* pSrc1,
    int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp<dstDatatype>* pQualityIndex);
```

Supported values for *mod*:

```
8u32f_C1R    32f_C1R
```

#### Case 2: Operation on multi-channel data

```
IppStatus ippiQualityIndex_<mod>(const Ipp<srcDatatype>* pSrc1,
    int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step,
    IppiSize roiSize, Ipp<dstDatatype> qualityIndex[3]);
```

Supported values for *mod*:

8u32f\_C3R    32f\_C3R  
8u32f\_AC4R   32f\_AC4R

## Parameters

<i>pSrc1</i> , <i>pSrc2</i>	Pointers to the source images ROI.
<i>src1Step</i> , <i>src2Step</i>	Distance in bytes between starts of consecutive lines in the source images.
<i>roiSize</i>	Size of the source ROI in pixels.
<i>pQualityIndex</i>	Pointer to the computed quality index value.
<i>qualityIndex</i>	An array containing the computed quality index values for separate channels in case of multi-channel data.

## Description

The function `ippiQualityIndex` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the universal image quality index for two images *pSrc1* and *pSrc2* according to the formula in the introduction section above. The computed value of the index for one-channel image is stored in *pQualityIndex*. For multi-channel images, the quality index is computed separately for each channel and stored in the array *qualityIndex*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>src1Step</i> or <i>src2Step</i> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## Image Proximity Measures

The functions described in this section compute the proximity (similarity) measure between an image and a template (another image). These functions may be used as feature detection functions, as well as the components of more sophisticated techniques.

There are several ways to compute the measure of similarity between two images. One way is to compute the Euclidean distance, or sum of the squared distances (SSD), of an image and a template. The smaller is the value of SSD at a particular pixel, the more similarity exists between the template and the image in the neighborhood of that pixel.

The squared Euclidean distance  $S_{tx}(r,c)$  between a template and an image for the pixel in row  $r$  and column  $c$  is given by the equation:

$$S_{tx}(r, c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} \left[ t(j, i) - x\left(r + j - \frac{tplRows}{2}, c + i - \frac{tplCols}{2}\right) \right]^2$$

where  $x(r,c)$  is the image pixel value in row  $r$  and column  $c$ , and  $t(j,i)$  is the template pixel value in row  $j$  and column  $i$ ; template size is  $tplCols$  by  $tplRows$  and its center is positioned at  $(r,c)$ .

The other similarity measure is the cross-correlation function: the higher is the cross-correlation at a particular pixel, the more similarity exists between the template and the image in the neighborhood of that pixel.

The cross-correlation  $R_{tx}(r,c)$  between a template and an image at the pixel in row  $r$  and column  $c$  is computed by the equation :

$$R_{tx}(r, c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t(j, i) \cdot x\left(r + j - \frac{tplRows}{2}, c + i - \frac{tplCols}{2}\right)$$

The cross-correlation function is dependent on the brightness variation across the image. To avoid this dependence, the correlation coefficient function is used instead. It is defined as:

$$G_{tx}(r, c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} [t(j, i) - \bar{t}] \cdot \left[ x\left(r + j - \frac{tplRows}{2}, c + i - \frac{tplCols}{2}\right) - \bar{x}(r, c) \right]$$



where  $\bar{x}$  is the mean of the template, and  $\bar{x}$  is the mean of the image in the region just under the template.

All Intel IPP proximity functions compute **normalized** values of SSD, cross-correlation and correlation coefficient that are defined as follows:

normalized SSD:  $\sigma_{tx}(r, c)$

$$\sigma_{tx}(r, c) = \frac{S_{tx}(r, c)}{\sqrt{R_{xx}(r, c)R_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right)}}$$

normalized cross-correlation  $\rho_{tx}(r, c)$  :

$$\rho_{tx}(r, c) = \frac{R_{tx}(r, c)}{\sqrt{R_{xx}(r, c)R_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right)}}$$

Here  $R_{xx}$  and  $R_{tt}$  denote the auto-correlation of the image and the template, respectively:

$$R_{xx}(r, c) = \sum_{j=r-\frac{tplRows-1}{2}}^{r+\frac{tplRows-1}{2}} \sum_{i=c-\frac{tplCols-1}{2}}^{c+\frac{tplCols-1}{2}} x_{j,i}x_{j,i}$$

$$R_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t_{j,i}t_{j,i}$$

Normalized correlation coefficient  $\gamma_{tx}(r, c)$  :

$$\gamma_{tx}(r, c) = \frac{G_{tx}(r, c)}{\sqrt{G_{xx}(r, c)G_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right)}}$$

Here  $G_{xx}$  and  $G_{tt}$  denote the auto-correlations of the image and the template without constant brightness component, respectively:

$$G_{xx}(r, c) = \sum_{j = r - \frac{tplRows-1}{2}}^{r + \frac{tplRows-1}{2}} \sum_{i = c - \frac{tplCols-1}{2}}^{c + \frac{tplCols-1}{2}} [x_{j,i} - \bar{x}(r, c)]^2$$

$$G_{tt}\left(\frac{tplRows}{2}, \frac{tplCols}{2}\right) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} (t_{j,i} - \bar{t})^2$$

---

## SqrDistanceFull\_Norm

*Computes normalized Euclidean distance between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output.

```
IppStatus ippiSqrDistanceFull_Norm_<mod>(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize,
    Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod* :

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

#### Case 2: Operation on data with floating-point output

```
IppStatus ippiSqrDistanceFull_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
    int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod*:

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiSqrDistanceFull_Norm` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes and returns the sum of squared distances (SSD) between the source and template images, that is, this function computes the normalized SSD value  $\sigma_{tx}(r, c)$  for each pixel in source and template buffers and stores the computed value in the corresponding pixel of the output image. The size of the resulting matrix with normalized SSD coefficients is

$$(W_s + W_t - 1) * (H_s + H_t - 1),$$

where  $W_s, H_s$  denote the width and height of the source image, and  $W_t, H_t$  denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\sigma_{tx}(r, c)$  in the introduction section above).

There are no any requirements for data outside the ROI, and the function assumes that template and source images are zero padded.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition when <i>srcRoiSize</i> or <i>tplRoiSize</i> has a field with zero or negative value, or when <i>srcRoiSize</i> has a field with value smaller than value of the corresponding field of <i>tplRoiSize</i> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <i>srcStep</i> , <i>tplStep</i> , or <i>dstStep</i> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## SqrDistanceSame\_Norm

*Computes normalized Euclidean distance between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```
ippStatus ippisSqrDistanceSame_Norm_<mod>(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

#### Case 2: Operation on data with floating-point output

```
ippStatus ippisSqrDistanceSame_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
    int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod*:

```
32f_C1R      8u32f_C1R      8s32f_C1R
32f_C3R      8u32f_C3R      8s32f_C3R
```

32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiSqrDistanceSame_Norm` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the normalized SSD values  $\sigma_{tx}(r, c)$  for the pixels that belong only to the source image. The size of the resulting matrix with normalized SSD coefficients is equal to the size of the source image:

$$W_s * H_s,$$

where  $W_s, H_s$  denote the width and height of the source image, respectively.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $r, c_{space}$  in the introduction section above).

There are no any requirements for data outside the ROI, and the function assumes that template and source images are zero padded.

## Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippiStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.

<code>ippStsSizeErr</code>	Indicates an error condition when <i>srcRoiSize</i> or <i>tplRoiSize</i> has a field with zero or negative value, or when <i>srcRoiSize</i> has a field with value smaller than value of the corresponding field of <i>tplRoiSize</i> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <i>srcStep</i> , <i>tplStep</i> , or <i>dstStep</i> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## SqrDistanceValid\_Norm

*Computes normalized Euclidean distance between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```
IppStatus ippisqrDistanceValid_Norm_<mod>(const Ipp8u* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

#### Case 2: Operation on data with floating-point output

```
IppStatus ippisqrDistanceValid_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int
    tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod*:

<code>32f_C1R</code>	<code>8u32f_C1R</code>	<code>8s32f_C1R</code>
<code>32f_C3R</code>	<code>8u32f_C3R</code>	<code>8s32f_C3R</code>
<code>32f_C4R</code>	<code>8u32f_C4R</code>	<code>8s32f_C4R</code>
<code>32f_AC4R</code>	<code>8u32f_AC4R</code>	<code>8s32f_AC4R</code>

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.
<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiSqrDistanceValid_Norm` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the normalized SSD values  $\sigma_{tx}(r, c)$  for the pixels of the source image without zero-padded edges. The size of the resulting matrix with normalized SSD values is

$$(W_s - W_t + 1) * (H_s - H_t + 1),$$

where  $W_s, H_s$  denote the width and height of the source image, and  $W_t, H_t$  denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\sigma_{tx}(r, c)$  in the introduction section above).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition when <i>srcRoiSize</i> or <i>tplRoiSize</i> has a field with zero or negative value, or when <i>srcRoiSize</i> has a field with value smaller than value of the corresponding field of <i>tplRoiSize</i> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <i>srcStep</i> , <i>tplStep</i> , or <i>dstStep</i> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## CrossCorrFull\_Norm

*Computes normalized cross-correlation  
between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrFull_Norm_<mod>(const Ipp8u* pSrc, int srcStep,  
    IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,  
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1RSfs  
8u_C3RSfs  
8u_C4RSfs  
8u_AC4RSfs
```

#### Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrFull_Norm_<mod>(const Ipp<srcDatatype>* pSrc,  
    int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,  
    int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod*:

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.



<code>tplRoiSize</code>	Size of the template ROI in pixels.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>scaleFactor</code>	The factor for integer result scaling.

## Description

The function `ippiCrossCorrFull_Norm` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes and returns the cross correlation between the source and template images, that is, this function computes the normalized cross-correlation value  $\rho_{tx}(x, c)$  for each pixel in source and template buffers and stores the computed value in the corresponding pixel of the output image. The size of the resulting matrix with normalized cross-correlation coefficients is

$$(W_s + W_t - 1) * (H_s + H_t - 1),$$

where  $W_s, H_s$  denote the width and height of the source image, and  $W_t, H_t$  denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\rho_{tx}(x, c)$  in the introduction section above.)

There are no any requirements for data outside the ROI, and the function assumes that template and source images are zero padded.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition when <code>srcRoiSize</code> or <code>tplRoiSize</code> has a field with zero or negative value, or when <code>srcRoiSize</code> has a field with value smaller than value of the corresponding field of <code>tplRoiSize</code> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <code>srcStep</code> , <code>tplStep</code> , or <code>dstStep</code> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## CrossCorrSame\_Norm

*Computes normalized cross-correlation between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrSame_Norm_<mod>(const Ipp8u* pSrc, int srcStep,  
    IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,  
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1RSfs  
8u_C3RSfs  
8u_C4RSfs  
8u_AC4RSfs
```

#### Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrSame_Norm_<mod>(const Ipp<srcDatatype>* pSrc,  
    int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,  
    int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod*:

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.

<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiCrossCorrSame_Norm` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the normalized cross-correlation values  $\rho_{tx}(r, c)$  for the pixels that belong only to the source image. The size of the resulting matrix with normalized cross-correlation values is equal to the size of the source image:

$$W_s * H_s,$$

where  $W_s, H_s$  denote the width and height of the source image, respectively.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\rho_{tx}(r, c)$  in the introduction section above.)

There are no any requirements for data outside the ROI, and the function assumes that template and source images are zero padded.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition when <i>srcRoiSize</i> or <i>tplRoiSize</i> has a field with zero or negative value, or when <i>srcRoiSize</i> has a field with value smaller than value of the corresponding field of <i>tplRoiSize</i> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <i>srcStep</i> , <i>tplStep</i> , or <i>dstStep</i> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## CrossCorrValid\_Norm

*Computes normalized cross-correlation between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrValid_Norm_<mod>(const Ipp8u* pSrc, int srcStep,
    IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod*:

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

#### Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrValid_Norm_<mod>(const Ipp<srcDatatype>* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int
    tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod*:

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.

<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiCrossCorrValid_Norm` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the normalized cross-correlation values  $\rho_{tx}(r, c)$  for the pixels of the source image without zero-padded edges. The size of the resulting matrix with normalized cross-correlation values is

$$(W_s - W_t + 1) * (H_s - H_t + 1),$$

where  $W_s, H_s$  denote the width and height of the source image, and  $W_t, H_t$  denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\rho_{tx}(r, c)$  in the introduction section above.)

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition when <i>srcRoiSize</i> or <i>tplRoiSize</i> has a field with zero or negative value, or when <i>srcRoiSize</i> has a field with value smaller than value of the corresponding field of <i>tplRoiSize</i> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <i>srcStep</i> , <i>tplStep</i> , or <i>dstStep</i> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## CrossCorrFull\_NormLevel

*Computes normalized correlation coefficient  
between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrFull_NormLevel_<mod>(const Ipp8u* pSrc,  
    int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,  
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod* :

```
8u_C1RSfs  
8u_C3RSfs  
8u_C4RSfs  
8u_AC4RSfs
```

#### Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrFull_NormLevel_<mod>(const Ipp<srcDatatype>*  
    pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,  
    int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* :

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.

<code>tplRoiSize</code>	Size of the template ROI in pixels.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>scaleFactor</code>	The factor for integer result scaling.

## Description

The function `ippiCrossCorrFull_NormLevel` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes and returns the correlation coefficient between the source and template images, that is, this function computes the normalized correlation coefficient  $\gamma_{tx}(x, c)$  for each pixel in source and template buffers and stores the computed value in the corresponding pixel of the output image. The size of the resulting matrix with normalized cross-correlation coefficients is

$$(W_s + W_t - 1) * (H_s + H_t - 1),$$

where  $W_s, H_s$  denote the width and height of the source image, and  $W_t, H_t$  denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\gamma_{tx}(x, c)$  in the introduction section above.)

There are no any requirements for data outside the ROI, and the function assumes that template and source images are zero padded.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition when <code>srcRoiSize</code> or <code>tplRoiSize</code> has a field with zero or negative value, or when <code>srcRoiSize</code> has a field with value smaller than value of the corresponding field of <code>tplRoiSize</code> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <code>srcStep</code> , <code>tplStep</code> , or <code>dstStep</code> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## CrossCorrSame\_NormLevel

*Computes normalized correlation coefficient between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrSame_NormLevel_<mod>(const Ipp8u* pSrc,  
    int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,  
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod* :

```
8u_C1RSfs  
8u_C3RSfs  
8u_C4RSfs  
8u_AC4RSfs
```

#### Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrSame_NormLevel_<mod>(const Ipp<srcDatatype>*  
    pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,  
    int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* :

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.



<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiCrossCorrSame_NormLevel` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the normalized correlation coefficients  $\gamma_{tx}(r, c)$  for the pixels that belong only to the source image. The size of the resulting matrix with normalized correlation coefficients is equal to the size of the source image:

$$W_s * H_s,$$

where  $W_s, H_s$  denote the width and height of the source image, respectively.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\gamma_{tx}(r, c)$  in the introduction section above.)

There are no any requirements for data outside the ROI, and the function assumes that template and source images are zero padded.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition when <i>srcRoiSize</i> or <i>tplRoiSize</i> has a field with zero or negative value, or when <i>srcRoiSize</i> has a field with value smaller than value of the corresponding field of <i>tplRoiSize</i> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <i>srcStep</i> , <i>tplStep</i> , or <i>dstStep</i> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

## CrossCorrValid\_NormLevel

*Computes normalized cross-correlation between an image and a template.*

---

### Syntax

#### Case 1: Operation with integer output

```
IppStatus ippiCrossCorrValid_NormLevel_<mod>(const Ipp8u* pSrc,
    int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep,
    IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor);
```

Supported values for *mod* :

```
8u_C1RSfs
8u_C3RSfs
8u_C4RSfs
8u_AC4RSfs
```

#### Case 2: Operation on data with floating-point output

```
IppStatus ippiCrossCorrValid_NormLevel_<mod>(const Ipp<srcDatatype>*
    pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl,
    int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep);
```

Supported values for *mod* :

32f_C1R	8u32f_C1R	8s32f_C1R
32f_C3R	8u32f_C3R	8s32f_C3R
32f_C4R	8u32f_C4R	8s32f_C4R
32f_AC4R	8u32f_AC4R	8s32f_AC4R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source ROI in pixels.
<i>pTpl</i>	Pointer to the template image buffer.
<i>tplStep</i>	Distance in bytes between starts of consecutive lines in the template image.

<i>tplRoiSize</i>	Size of the template ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiCrossCorrValid_NormLevel` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function computes the normalized correlation coefficients  $\gamma_{tx}(r, c)$  for the pixels of the source image without zero-padded edges. The size of the resulting matrix with normalized correlation coefficients is

$$(W_s - W_t + 1) * (H_s - H_t + 1),$$

where  $W_s, H_s$  denote the width and height of the source image, and  $W_t, H_t$  denote the width and height of the template.

The template anchor for matching the image pixel is always at the geometric center of the template. (See the formula for  $\gamma_{tx}(r, c)$  in the introduction section above.)

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition when <i>srcRoiSize</i> or <i>tplRoiSize</i> has a field with zero or negative value, or when <i>srcRoiSize</i> has a field with value smaller than value of the corresponding field of <i>tplRoiSize</i> .
<code>ippStsStepErr</code>	Indicates an error condition if at least one of <i>srcStep</i> , <i>tplStep</i> , or <i>dstStep</i> has a zero or negative value.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

# Image Geometric Transforms

# 12

This chapter describes Intel IPP functions that perform geometric operations of resizing, rotating, warping and remapping an image.

[Table 12-1](#) lists the Intel IPP image geometric transform functions:

**Table 12-1 Image Geometric Transform Functions**

Function Base Name	Operation
<a href="#">Resize</a>	Changes the size of an image.
<a href="#">ResizeCenter</a>	Changes the size of an image and shifts it.
<a href="#">ResizeSqrPixel</a>	Changes the size of an image using different interpolation algorithm.
<a href="#">ResizeSqrPixelGetBufSize</a>	Computes the size of the external buffer for image resizing.
<a href="#">GetResizeFract</a>	Recalculates resize factors for a tiled image.
<a href="#">ResizeShift</a>	Changes the size of an image tile.
<a href="#">ResizeYUV422</a>	Changes the size of an 2-channel image in 4:2:2 sampling format.
<a href="#">Mirror</a>	Mirrors an image.
<a href="#">Remap</a>	Performs arbitrary remapping of pixels in an image.
<a href="#">Rotate</a>	Rotates an image around the origin (0,0) and shifts the result.
<a href="#">GetRotateShift</a>	Computes the shift values for <code>ippiRotate</code> , given the rotation center and angle.
<a href="#">AddRotateShift</a>	Computes shift values for an image rotation around the specified center with specified shifts.
<a href="#">GetRotateQuad</a>	Computes the quadrangle that results from an image ROI transformed by <code>ippiRotate</code> function.

**Table 12-1 Image Geometric Transform Functions (continued)**

Function Base Name	Operation
<a href="#"><u>GetRotateBound</u></a>	Computes the bounding rectangle for an image ROI transformed by <code>ippiRotate</code> function.
<a href="#"><u>RotateCenter</u></a>	Rotates an image around an arbitrary center.
<a href="#"><u>Shear</u></a>	Performs a shearing transform of an image.
<a href="#"><u>GetShearQuad</u></a>	Computes the quadrangle that results from an image ROI transformed by <code>ippiShear</code> function.
<a href="#"><u>GetShearBound</u></a>	Computes the bounding rectangle for an image ROI transformed by <code>ippiShear</code> function.
<a href="#"><u>WarpAffine</u></a>	Warpes an image with an affine transform.
<a href="#"><u>WarpAffineBack</u></a>	Performs an inverse affine transform of an image.
<a href="#"><u>WarpAffineQuad</u></a>	Performs an affine transform of an image from the source quadrangle to the destination quadrangle.
<a href="#"><u>GetAffineQuad</u></a>	Computes the quadrangle that results from an image ROI transformed by <code>ippiWarpAffine</code> function.
<a href="#"><u>GetAffineBound</u></a>	Computes the bounding rectangle for an image ROI transformed by <code>ippiWarpAffine</code> function.
<a href="#"><u>GetAffineTransform</u></a>	Computes the affine transform coefficients.
<a href="#"><u>WarpPerspective</u></a>	Warpes an image with a perspective transform.
<a href="#"><u>WarpPerspectiveBack</u></a>	Performs an inverse perspective transform of an image.
<a href="#"><u>WarpPerspectiveQuad</u></a>	Performs a perspective transform of an image from the source quadrangle to the destination quadrangle.
<a href="#"><u>GetPerspectiveQuad</u></a>	Computes the quadrangle that results from an image ROI transformed by <code>ippiWarpPerspective</code> function.
<a href="#"><u>GetPerspectiveBound</u></a>	Computes the bounding rectangle for an image ROI transformed by <code>ippiWarpPerspective</code> function.
<a href="#"><u>GetPerspectiveTransform</u></a>	Computes the perspective transform coefficients.
<a href="#"><u>WarpBilinear</u></a>	Warpes an image with a bilinear transform.
<a href="#"><u>WarpBilinearBack</u></a>	Performs an inverse bilinear transform of an image.
<a href="#"><u>WarpBilinearQuad</u></a>	Performs a bilinear transform of an image from the source quadrangle to the destination quadrangle.
<a href="#"><u>GetBilinearQuad</u></a>	Computes the quadrangle that results from an image ROI transformed by <code>ippiWarpBilinear</code> function.
<a href="#"><u>GetBilinearBound</u></a>	Computes the bounding rectangle for an image ROI transformed by <code>ippiWarpBilinear</code> function.

**Table 12-1** Image Geometric Transform Functions (continued)

Function Base Name	Operation
<a href="#">GetBilinearTransform</a>	Computes the bilinear transform coefficients.

Each function performing geometric transform of an image uses some interpolation algorithm to resample the image. The type of interpolation method to be used is passed to the function in the *interpolation* parameter.

The following interpolation algorithms are used:

- nearest neighbor
- linear interpolation
- cubic convolution
- supersampling
- interpolation using Lanczos window function
- optional edge smoothing of the destination image.

The nearest neighbor algorithm is the fastest, while other methods yield higher quality results, but are slower.

Use one of the following constant identifiers for the applicable interpolation methods:

IPPI_INTER_NN	Nearest neighbor interpolation.
IPPI_INTER_LINEAR	Linear interpolation
IPPI_INTER_CUBIC	Cubic interpolation
IPPI_INTER_LANCZOS	Interpolation using 3-lobed Lanczos window function.
IPPI_INTER_SUPER	Supersampling interpolation.

For certain functions, you can combine the above interpolation algorithms with additional smoothing (antialiasing) of edges to which the original image borders are transformed. To use this edge smoothing, set the parameter *interpolation* to the bitwise OR of `IPPI_SMOOTH_EDGE` and the desired interpolation mode.



---

**CAUTION.** You can use interpolation with edge smoothing option only in those geometric transform functions where this option is explicitly listed in the parameters definition section.

---

See appendix B “[Interpolation in Image Geometric Transform Functions](#)” for more information on the interpolation algorithms that are used in the library.

Many solutions and hints for use of these functions can be found in Intel® IPP Samples. See *Intel IPP Image Processing and Media Processing Samples* downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>.

## ROI Processing in Geometric Transforms

All the transform functions described in this chapter operate in rectangular regions of interest (ROIs) that are defined in both the source and destination images.

The procedures for handling ROIs in geometric transform functions differ from those used in other functions (see [Regions of Interest in IPP](#) in chapter 2). The main difference is that operations take place in the intersection of the *transformed* source ROI and the destination ROI. More specifically, all geometric transform functions (except those which perform inverse warping operations) handle ROIs with the following sequence of operations:

- transform the rectangular ROI of the source image to quadrangle in the destination image;
- find the intersection of this quadrangle and the rectangular ROI of the destination image;
- update the destination image in the intersection area.

The coordinates in the source and destination images must have the same origin.

Using functions with ROI allows every scanline of a source image to be padded with zeroes for alignment, so that actual size of a scanline in bytes can be greater than it may be assumed from the image width. In that case the size of each row of image data in bytes is determined by the value of *srcStep* parameter, which gives distance in bytes between the starts of consecutive lines of an image.

To fully describe a rectangular ROI, both its origin (coordinates of top left corner) and size must be referenced. For geometrical transform functions, the source image ROI is specified by *srcRoi* parameter of *IppiRect* type, meaning that all four values describing the rectangular ROI are given explicitly.

On the other hand, the destination image ROI for different functions can be specified either by *dstRoi* parameter of *IppiRect* type, or *dstRoiSize* parameter of *IppiSize* type. In the latter case, only the destination ROI size is passed, while its origin is referenced by *pDst* pointer.

---

## Resize

*Changes an image size.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiResize_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep,
    IppiSize dstRoiSize, double xFactor, double yFactor,
    int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippiResize_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[3], int dstStep, IppiSize dstRoiSize,
    double xFactor, double yFactor, int interpolation);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------

```
IppStatus ippiResize_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[4], int dstStep, IppiSize dstRoiSize,
    double xFactor, double yFactor, int interpolation);
```



Supported values for *mod*:

8u\_P4R          16u\_P4R          32f\_P4R

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.										
<i>srcSize</i>	Size in pixels of the source image.										
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.										
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).										
<i>pDst</i>	Pointer to the destination image ROI. An array of pointers to separate ROI in each planes for planar image.										
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.										
<i>dstRoiSize</i>	Size of the destination ROI in pixels.										
<i>xFactor, yFactor</i>	Factors by which the <i>x</i> and <i>y</i> dimensions of the source ROI are changed. The factor value greater than 1 corresponds to increasing the image size in that dimension.										
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for image resampling. Use one of the following values: <table> <tr> <td>IPPI_INTER_NN</td><td>nearest neighbor interpolation</td></tr> <tr> <td>IPPI_INTER_LINEAR</td><td>linear interpolation</td></tr> <tr> <td>IPPI_INTER_CUBIC</td><td>cubic interpolation</td></tr> <tr> <td>IPPI_INTER_SUPER</td><td>supersampling interpolation, can not be applied for image enlarging</td></tr> <tr> <td>IPPI_INTER_LANCZOS</td><td>interpolation with Lanczos window.</td></tr> </table>	IPPI_INTER_NN	nearest neighbor interpolation	IPPI_INTER_LINEAR	linear interpolation	IPPI_INTER_CUBIC	cubic interpolation	IPPI_INTER_SUPER	supersampling interpolation, can not be applied for image enlarging	IPPI_INTER_LANCZOS	interpolation with Lanczos window.
IPPI_INTER_NN	nearest neighbor interpolation										
IPPI_INTER_LINEAR	linear interpolation										
IPPI_INTER_CUBIC	cubic interpolation										
IPPI_INTER_SUPER	supersampling interpolation, can not be applied for image enlarging										
IPPI_INTER_LANCZOS	interpolation with Lanczos window.										

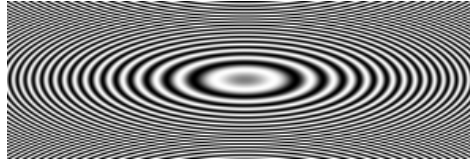
## Description

The function `ippiResize` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function resizes the source image ROI by *xFactor* in the *x* direction and *yFactor* in the *y* direction. The image size can be either reduced or increased in each direction, depending on the values of *xFactor, yFactor*. The result is resampled using the [interpolation method](#) specified

by the *interpolation* parameter, and written to the destination image ROI. Supersampling interpolation can be applied if both *xFactor* and *yFactor* are less than or equal to 1. [Figure 12-1](#) shows the result of applying the `ippiResize` function to the [sample image](#) of size 256 by 256 pixels. The destination image is of 300 by 100 pixels size.

**Figure 12-1 Sample Image Changed by the `ippiResize` Function**



The following code [Example 12-1](#) shows how to use the `ippiResize` function:

**Example 12-1 Resizing an Image**

```
IppStatus resize( void ) {
    Ipp8u x[8*8], y[8*8];
    IppiSize srcsz={8,8}, dstroi={6,6};
    IppiRect srcroi={1,1,6,6};
    IppiSize roi = {8,8};
    int i;
    for( i=0; i<8; ++i ) {
        ippiSet_8u_C1R( (Ipp8u)i, x+8*i+i, 8, roi );
        --roi.width;
        --roi.height;
    }
    return ippiResize_8u_C1R( x, srcsz, 8, srcroi, y, 8, dstroi,
        2.9, 2, IPPI_INTER_NN );
}
```

The image has the following contents after resizing:

```
01 01 01 01 01 01 CC CC
01 01 01 01 01 01 CC CC
01 01 02 02 02 02 CC CC
01 01 02 02 02 02 CC CC
01 01 02 02 02 03 CC CC
01 01 02 02 02 03 CC CC
CC CC CC CC CC CC CC CC
CC CC CC CC CC CC CC CC
```

This function is used in H.264 encoder included into Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsResizeNoOperationErr</code>	Indicates an error condition if one of the destination image dimensions is less than 1 pixel.
<code>ippStsResizeFactorErr</code>	Indicates an error condition if <i>xFactor</i> or <i>yFactor</i> is less than or equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.

---

## ResizeCenter

*Changes an image size and shifts the destination image relative to the given point.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiResizeCenter_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,
    int dstStep, IppiSize dstRoiSize, double xFactor, double yFactor,
    double xCenter, double yCenter, int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

```
8u_C4R      16u_C4R      32f_C4R
8u_AC4R     16u_AC4R     32f_AC4R
```

## Case 2: Operation on planar-order data

```
IppStatus ippiResizeCenter_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[3], int dstStep, IppiSize dstRoiSize,
    double xFactor, double yFactor, double xCenter, double yCenter,
    int interpolation);
```

Supported values for *mod*:

```
8u_P3R      16u_P3R      32f_P3R
```

```
IppStatus ippiResizeCenter_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[4], int dstStep, IppiSize dstRoiSize,
    double xFactor, double yFactor, double xCenter, double yCenter,
    int interpolation);
```

Supported values for *mod*:

```
8u_P4R      16u_P4R      32f_P4R
```

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of 3 or 4 pointers to different color planes in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>pDst</i>	Pointer to the destination image ROI. An array of 3 or 4 pointers to destination ROI in different color planes for planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>xFactor</i> , <i>yFactor</i>	Factors by which the <i>x</i> and <i>y</i> dimensions of the source ROI are changed. The factor value greater than 1 corresponds to increasing the image size in that dimension.

<i>xCenter, yCenter</i>	Coordinates of the preset center.
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for image resampling. Use one of the following values:
IPPI_INTER_NN	nearest neighbor interpolation
IPPI_INTER_LINEAR	linear interpolation
IPPI_INTER_CUBIC	cubic interpolation
IPPI_INTER_SUPER	supersampling interpolation, can not be applied for image enlarging
IPPI_INTER_LANCZOS	interpolation with Lanczos window.

## Description

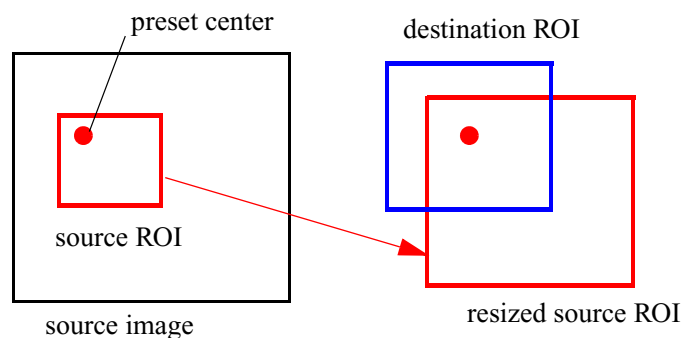
The function `ippiResizeCenter` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function resizes the source image ROI by *xFactor* in the *x* direction and *yFactor* in the *y* direction. The image size can be either reduced or increased in each direction, depending on the values of *xFactor*, *yFactor*.

Unlike `ippiResize`, this function also shifts the destination image in such a way that a given point with coordinate *xCenter*, *yCenter* - preset center - remains stationary after resizing, which creates an effect of changing the image size relative to that point. The preset center is always positioned in the center of the destination image ROI with coordinates `dstROI.width/2`; `dstROI.height/2` (see [Figure 12-2](#)). Note that the preset center may be specified in an arbitrary point including points outside the source image..

**Figure 12-2 Resizing Image with the Function `ippiResizeCenter`**

---



The result is resampled using the [interpolation method](#) specified by the *interpolation* parameter, and written to the destination image ROI. Supersampling interpolation can be applied if both *xFactor* and *yFactor* are less than or equal to 1.

The code [Example 12-2](#) shows how to use the `ippiResizeCenter` function.

### Example 12-2 Resizing and Shifting an Image

```
IppStatus resizeCenter( void ) {
    Ipp8u x[8*8], y[8*8];
    IppiSize sz = {8,8}, roi = {8,8};
    IppiRect rect = {0,0,8,8};
    int i;
    for( i=0; i<8; ++i ) {
        ippiSet_8u_C1R( (Ipp8u)i, x+8*i+i, 8, roi );
        --roi.width;
        --roi.height;
    }
    return ippiResizeCenter_8u_C1R( x, sz, 8, rect, y, 8, sz,
        2.9, 2.0, 4, 4, IPPI_INTER_NN );
}
```

The image has the following contents after applying the `ippiResizeCenter` function:

```
02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 02
02 02 03 03 03 03 03 03
02 02 03 03 03 03 03 03
02 02 03 03 03 04 04 04
02 02 03 03 03 04 04 04
02 02 03 03 03 04 04 04
02 02 03 03 03 04 04 04
```

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.

<code>ippStsWrongIntersectROI</code>	Indicates a warning if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsResizeNoOperationErr</code>	Indicates an error condition if one of the destination image dimensions is less than 1 pixel.
<code>ippStsResizeFactorErr</code>	Indicates an error condition if <i>xFactor</i> or <i>yFactor</i> is less than or equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.

---

## ResizeSqrPixel

*Changes an image size using different interpolation algorithm.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippResizeSqrPixel_<mod>(const Ipp<datatype>* pSrc, IppiSize
srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep,
IppiRect dstRoi, double xFactor, double yFactor, double xShift, double
yShift, int interpolation, Ipp8u* pBuffer);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippResizeSqrPixel_<mod>(const Ipp<datatype>* const pSrc[3],
IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
pDst[3], int dstStep, IppiRect dstRoi, double xFactor, double
yFactor, double xShift, double yShift, int interpolation, Ipp8u*
pBuffer);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------

```
IppStatus ippiResizeSqrPixel_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, double xFactor, double
    yFactor, double xShift, double yShift, int interpolation, Ipp8u*
    pBuffer);
```

Supported values for *mod*:

8u\_P4R          16u\_P4R          32f\_P4R

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of 3 or 4 pointers to different color planes for planar image.						
<i>srcSize</i>	Size in pixels of the source image.						
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.						
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).						
<i>pDst</i>	Pointer to the destination image. An array of 3 or 4 pointers to the separate color planes for planar image.						
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.						
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).						
<i>xFactor</i> , <i>yFactor</i>	Factors by which the <i>x</i> and <i>y</i> dimensions of the source ROI are changed. The factor value greater than 1 corresponds to increasing the image size in that dimension.						
<i>xShift</i> , <i>yShift</i>	Shift values in the <i>x</i> and <i>y</i> directions respectively.						
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for image resampling. Use one of the following values: <table data-bbox="617 1221 1299 1340"> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation						
<code>IPPI_INTER_LINEAR</code>	linear interpolation						
<code>IPPI_INTER_CUBIC</code>	cubic interpolation						
<i>pBuffer</i>	Pointer to the external buffer.						



## Description

The function `ippiResizeSqrPixel` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function resizes the source image ROI by *xFactor* in the *x* direction and *yFactor* in the *y* direction. The image size can be either reduced or increased in each direction, depending on the values of *xFactor*, *yFactor*. The result is resampled using the [interpolation method](#) specified by the *interpolation* parameter, and written to the destination image ROI.

Unlike `ippiResize`, this function uses different algorithm for interpolation.

Pixel coordinates  $x'$  and  $y'$  in the resized image are obtained from the following equations:

$$\begin{aligned}x' &= xFactor * x + xShift \\ y' &= yFactor * y + yShift\end{aligned}$$

where *x* and *y* denote the pixel coordinates in the source image.

The function requires the external buffer *pBuffer*, its size can be previously computed by calling the function [ippiResizeSqrPixelGetBufSize](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if width or height of images has zero or negative value.
<code>ippStsResizeFactorErr</code>	Indicates an error condition if <i>xFactor</i> or <i>yFactor</i> is less than or equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsWrongIntersectROI</code>	Indicates a warning if <i>srcRoi</i> has no intersection with the source image.

---

## ResizeSqrPixelGetBufSize

*Computes the size of the external buffer for image resizing.*

---

### Syntax

```
IppStatus ippiResizeSqrPixelGetBufSize(IppiSize dstSize, int nChannel,  
    int interpolation, int* pBufferSize);
```

### Parameters

<i>dstSize</i>	Size in pixels of the destination image.						
<i>nChannel</i>	Number of channels or planes, possible values are 1, 3, 4.						
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for image resampling. Use one of the following values: <div><table><tr><td>IPPI_INTER_NN</td><td>nearest neighbor interpolation</td></tr><tr><td>IPPI_INTER_LINEAR</td><td>linear interpolation</td></tr><tr><td>IPPI_INTER_CUBIC</td><td>cubic interpolation</td></tr></table></div>	IPPI_INTER_NN	nearest neighbor interpolation	IPPI_INTER_LINEAR	linear interpolation	IPPI_INTER_CUBIC	cubic interpolation
IPPI_INTER_NN	nearest neighbor interpolation						
IPPI_INTER_LINEAR	linear interpolation						
IPPI_INTER_CUBIC	cubic interpolation						
<i>pBufferSize</i>	Pointer to the size of the external buffer.						

### Description

The function `ippiResizeSqrPixelGetBufSize` is declared in the `ippi.h` file. This function computes the size of the external buffer that is required for the function [ippiResizeSqrPixel](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pBufferSize</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>dstSize</i> has a field with zero or negative value.
<code>ippStsNumChannelErr</code>	Indicates an error condition if <i>nChannel</i> has an illegal value.

<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
-------------------------------------	--

---

## GetResizeFract

*Recalculates resize factors for a tiled image.*

---

### Syntax

```
IppStatus ippiGetResizeFract(IppiSize srcSize, IppiRect srcRoi,
    double xFactor, double yFactor, double* xFr, double* yFr,
    int interpolation);
```

### Parameters

<i>srcSize</i>	Size of the source image in pixels.								
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).								
<i>xFactor, yFactor</i>	Factors by which the <i>x</i> and <i>y</i> dimensions of the source ROI are changed. The factor value greater than 1 corresponds to increasing the image size in that dimension.								
<i>xFr, yFr</i>	Pointers to the recalculated resize factors. These values should be passed to <code>ippiResizeShift</code> function to bring about the desired resizing of image tiles. The factor value greater than 1 corresponds to increasing the image size in that dimension.								
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for image resampling. Use one of the following values: <table border="0" style="margin-left: 40px;"> <tr> <td><code>IPPI_INTER_NN</code></td> <td>nearest neighbor interpolation</td> </tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td> <td>linear interpolation</td> </tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td> <td>cubic interpolation.</td> </tr> <tr> <td><code>IPPI_INTER_LANCZOS</code></td> <td>interpolation with Lanczos window.</td> </tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation.	<code>IPPI_INTER_LANCZOS</code>	interpolation with Lanczos window.
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation								
<code>IPPI_INTER_LINEAR</code>	linear interpolation								
<code>IPPI_INTER_CUBIC</code>	cubic interpolation.								
<code>IPPI_INTER_LANCZOS</code>	interpolation with Lanczos window.								

### Description

The function `ippiResizeFract` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

Use this function if you need to resize the tiled source image by *xFactor* in the *x* direction and *yFactor* in the *y* direction. It calculates the new values of resize factors *xFr* and *yFr* that should be passed to the [ippiResizeShift](#) function, which performs resize operation on the individual tiles.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> has a field with zero or negative value.
<code>ippStsResizeFactorErr</code>	Indicates an error condition if <i>xFactor</i> or <i>yFactor</i> is less than or equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.

---

## ResizeShift

*Changes the size of an image tile.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiResizeShift_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,
    int dstStep, IppiSize dstRoiSize, double xFr, double yFr,
    double xShift, double yShift, int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

**Case 2: Operation on planar-order data**

```
IppStatus ippiResizeShift_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[3], int dstStep, IppiSize dstRoiSize,
    double xFr, double yFr, double xShift, double yShift,
    int interpolation);
```

Supported values for *mod*:

8u\_P3R          16u\_P3R          32f\_P3R

```
IppStatus ippiResizeShift_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[4], int dstStep, IppiSize dstRoiSize,
    double xFr, double yFr, double xShift, double yShift,
    int interpolation);
```

Supported values for *mod*:

8u\_P4R          16u\_P4R          32f\_P4R

**Parameters**

<i>pSrc</i>	Pointer to the source image. An array of 3 or 4 pointers to different color planes in case of data in planar format.
<i>srcSize</i>	Size of the source image in pixels.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>pDst</i>	Pointer to the destination image ROI. An array of 3 or 4 pointers to destination ROI in different color planes for planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.
<i>xFr</i> , <i>yFr</i>	Factors, that are used to obtain the desired resizing of the source image ROI by the <i>xFactor</i> , <i>yFactor</i> .
<i>xShift</i> , <i>yShift</i>	Corrections of the position of the processing area ( <i>srcRect</i> ).

<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for image resampling. Use one of the following values:	
	IPPI_INTER_NN	nearest neighbor interpolation
	IPPI_INTER_LINEAR	linear interpolation
	IPPI_INTER_CUBIC	cubic interpolation.
	IPPI_INTER_LANCZOS	interpolation with Lanczos window.

## Description

The function `ippiResizeShift` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used for resizing the tiled image by *xFactor* in the *x* direction and *yFactor* in the *y* direction. Prior to using it, the `ippiGetResizeFract` function should be called to calculate the values of the *xFr* and *yFr* factors. The image size can be either reduced or increased in each direction, depending on the values of *xFactor*, *yFactor*.

The value of *xFr* or *yFr* factor greater than 1 corresponds to reducing the size, and the value less than 1 corresponds to increasing the image size in that dimension.

The result is resampled using the [interpolation method](#) specified by the *interpolation* parameter, and written to the destination image ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsResizeFactorErr</code>	Indicates an error condition if <i>xFr</i> or <i>yFr</i> is less than or equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.

---

## ResizeYUV422

*Changes a size of two-channel image.*

---

### Syntax

```
IppStatus ippiResizeYUV422_8u_C2R(const Ipp8u* pSrc, IppiSize srcSize,  
int srcStep, IppiRect srcRoi, Ipp8u* pDst, int dstStep, IppiSize  
dstRoiSize, double xFactor, double yFactor, int interpolation);
```

### Parameters

<i>pSrc</i>	Pointer to the source image data.								
<i>srcSize</i>	Size in pixels of the source image.								
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.								
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).								
<i>pDst</i>	Pointer to the destination ROI buffer.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.								
<i>dstRoiSize</i>	Size of the destination ROI in pixels.								
<i>xFactor, yFactor</i>	Factors by which the <i>x</i> and <i>y</i> dimensions of the source ROI are changed. The factor value greater than 1 corresponds to increasing the image size in that dimension.								
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for image resampling. Use one of the following values: <table><tr><td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr><tr><td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr><tr><td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr><tr><td><code>IPPI_INTER_SUPER</code></td><td>supersampling interpolation, can not be applied for image enlarging</td></tr></table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation	<code>IPPI_INTER_SUPER</code>	supersampling interpolation, can not be applied for image enlarging
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation								
<code>IPPI_INTER_LINEAR</code>	linear interpolation								
<code>IPPI_INTER_CUBIC</code>	cubic interpolation								
<code>IPPI_INTER_SUPER</code>	supersampling interpolation, can not be applied for image enlarging								

### Description

The function `ippiResizeYUV422` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function resizes the source image ROI by *xFactor* in the *x* direction and *yFactor* in the *y* direction. The image size can be either reduced or increased in each direction, depending on the values of *xFactor*, *yFactor*. The result is resampled using the [interpolation method](#) specified by the *interpolation* parameter, and written to the destination image ROI. Supersampling interpolation can be applied if both *xFactor* and *yFactor* are less than or equal to 1.

The source image is a two-channel image in 4:2:2 sampling format in color spaces with decoupled luminance and chrominance components, for example, YUV422 or YCrCb422.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> or <i>dstRoiSize</i> has a field with zero or negative value; or if <i>dstRoiSize.width</i> is less than 2; or if width of the source image or the source ROI is odd; or if horizontal offset of the source ROI is odd.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsResizeNoOperationErr</code>	Indicates an error condition if one of the destination image dimensions is less than 1 pixel.
<code>ippStsResizeFactorErr</code>	Indicates an error condition if <i>xFactor</i> or <i>yFactor</i> is less than or equal to zero.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.

---

## Mirror

*Mirrors an image about a horizontal or vertical axis, or both.*

---

### Syntax

#### Case 1: Not-in-place operation

```
IppStatus ippMirror_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiAxis flip);
```



Supported values for *mod*:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

### Case 2: In-place operation

```
IppStatus ippMirror_<mod>(const Ipp<datatype>* pSrcDst, int srcDstStep,
    IppiSize roiSize, IppiAxis flip);
```

Supported values for *mod*:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

### Parameters

<i>pSrc</i>	Pointer to the source buffer.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pDst</i>	Pointer to the destination buffer.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>pSrcDst</i>	Pointer to the source and destination buffer for the in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.
<i>flip</i>	Specifies the axis to mirror the image about. Use the following values to specify the axes: <div> <div>ippAxsHorizontal</div> <div>for the horizontal axis</div> <div>ippAxsVertical</div> <div>for the vertical axis</div> <div>ippAxsBoth</div> <div>for both horizontal and vertical axes.</div> </div>

## Description

The function `ippiMirror` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function mirrors the source image `pSrc` about a horizontal or vertical axis or both, depending on the `flip` value, and writes it to the destination image `pDst`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsMirrorFlipErr</code>	Indicates an error condition if <code>flip</code> has an illegal value.

---

## Remap

*Applies the look-up coordinate mapping to pixels of a source image.*

---

## Syntax

### Case 1: Operation on pixel-order data

```
IppStatus ippiRemap_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcRoi, const Ipp32f* pxMap, int xMapStep,
    const Ipp32f* pyMap, int yMapStep, Ipp<datatype>* pDst, int dstStep,
    IppiSize dstRoiSize, int interpolation);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>32f_AC4R</code>

**Case 2: Operation on planar-order data**

```
IppStatus ippiRemap_<mod>(const Ipp<datatype>* const pSrc[3],  
    IppiSize srcSize, int srcStep, IppiRect srcRoi, const Ipp32f* pxMap,  
    int xMapStep, const Ipp32f* pyMap, int yMapStep, Ipp<datatype>* const  
    pDst[3], int dstStep, IppiSize dstRoiSize, int interpolation);
```

Supported values for *mod*:

8u\_P3R            32f\_P3R

```
IppStatus ippiRemap_<mod>(const Ipp<datatype>* const pSrc[4],  
    IppiSize srcSize, int srcStep, IppiRect srcRoi, const Ipp32f* pxMap,  
    int xMapStep, const Ipp32f* pyMap, int yMapStep, Ipp<datatype>* const  
    pDst[4], int dstStep, IppiSize dstRoiSize, int interpolation);
```

Supported values for *mod*:

8u\_P4R            32f\_P4R

**Parameters**

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>pxMap</i> , <i>pyMap</i>	Pointers to the starts of 2D buffers, containing tables of the <i>x</i> - and <i>y</i> -coordinates. If the referenced coordinates correspond to a pixel outside of the source ROI, no mapping of the source pixel is done.
<i>xMapStep</i> , <i>yMapStep</i>	Steps in bytes through the buffers containing tables of the <i>x</i> - and <i>y</i> -coordinates.
<i>pDst</i>	Pointer to the destination image ROI. An array of separate pointers to ROI in each plane for planar image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoiSize</i>	Size of the destination ROI in pixels.

*interpolation*      The type of [interpolation](#) to perform for image resampling. Use one of the following values:

IPPI_INTER_NN	nearest neighbor interpolation
IPPI_INTER_LINEAR	linear interpolation
IPPI_INTER_CUBIC	cubic interpolation.

## Description

The function `ippiRemap` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function transforms the source image by remapping its pixels. Pixel remapping is performed using `pxMap` and `pyMap` buffers to look-up the coordinates of the source image pixel that is written to the target destination image pixel. Your application has to supply these look-up tables. The remapping of source pixels to destination pixels is made according to the following formula:

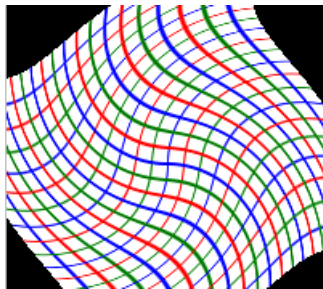
$$dst\_pixel[i, j] = src\_pixel[pxMap[i, j], pyMap[i, j]]$$

where  $i, j$  are the  $x$ - and  $y$ -coordinates of the target destination image pixel `dst_pixel`;  
`pxMap[i, j]` contains the  $x$ - coordinate of the source image pixel `src_pixel` that should be written to `dst_pixel`  
`pyMap[i, j]` contains the  $y$ - coordinate of the source image pixel `src_pixel` that should be written to `dst_pixel`

[Figure 12-3](#) gives an example of applying the function `ippiRemap` to a sample image that is a square grid of alternating blue, red, and green lines

**Figure 12-3    Remapping the Sample Image**

---



The transformed part of the image is resampled using the [interpolation method](#) specified by the *interpolation* parameter, and is written to the destination image ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.

---

## Rotate

*Rotates an image around the origin (0,0) and shifts it.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippIRotate_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep,
    IppiRect dstRoi, double angle, double xShift, double yShift,
    int interpolation);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>16u_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>16u_AC4R</code>	<code>32f_AC4R</code>

**Case 2: Operation on planar-order data**

```
IppStatus ippiRotate_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, double angle, double xShift,
    double yShift, int interpolation);
```

Supported values for *mod*:

8u\_P3R          16u\_P3R          32f\_P3R

```
IppStatus ippiRotate_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, double angle, double xShift,
    double yShift, int interpolation);
```

Supported values for *mod*:

8u\_P4R          16u\_P4R          32f\_P4R

**Parameters**

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).
<i>angle</i>	The angle of rotation in degrees. The source image is rotated counterclockwise around the origin (0,0).
<i>xShift, yShift</i>	The shifts along horizontal and vertical axes to perform after the rotation.
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for resampling the image. Use one of the following he following values: <div style="text-align: right;"> <code>IPPI_INTER_NN</code>          nearest neighbor interpolation </div>

IPPI_INTER_LINEAR	linear interpolation
IPPI_INTER_CUBIC	cubic interpolation
IPPI_SMOOTH_EDGE	use edge smoothing option in addition to one of the above methods.

## Description

The function `ippiRotate` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

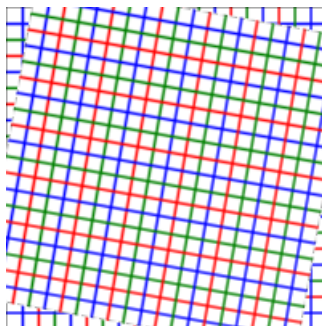
This function rotates the ROI of the source image by *angle* degrees (counterclockwise for positive *angle* values) around the origin (0,0) that is implied to be in the top left corner, and shifts it by *xShift* and *yShift* values along the *x*- and *y*- axes, respectively. The result is resampled using the [interpolation method](#) specified by the *interpolation* parameter, and written to the destination image ROI.

If you need to rotate an image about an arbitrary center (*xCenter*, *yCenter*), use the function [ippiGetRotateShift](#) to compute the appropriate *xShift*, *yShift* values, and then call [ippiRotate](#) with these values as input parameters. Alternatively, you can use the [ippiRotateCenter](#) function instead.

[Figure 12-4](#) shows the result of rotating the sample image by 10 degrees.

**Figure 12-4**    **Rotation of an Image**

---



The figures below illustrate how the `SMOOTH_EDGE` option affects the resulting destination image appearance. Both images are computed as a result of rotating the same source image by 9 degrees using the same [interpolation method](#), with the `SMOOTH_EDGE` option turned off and on, respectively. The use of this option has an antialiasing effect seen on the right border of image b).

**Figure 12-5 The Effect of Edge Smoothing in the Destination Image**



a) `SMOOTH_EDGE` option turned off



b) `SMOOTH_EDGE` option turned on

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the intersection of the <i>srcRoi</i> and source image is less than or equal to 1.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.



---

## GetRotateShift

*Computes shift values for rotation  
of an image around the specified center.*

---

### Syntax

```
IppStatus ippiGetRotateShift(double xCenter, double yCenter,  
                             double angle, double* xShift, double* yShift)
```

### Parameters

<code>xCenter, yCenter</code>	Coordinates of the required center of rotation.
<code>angle</code>	The angle in degrees to rotate the image clockwise around the point with coordinates ( <code>xCenter, yCenter</code> ).
<code>xShift, yShift</code>	Pointers to computed shift values along horizontal and vertical axes. These shift values should be passed to <code>ippiRotate</code> function to bring about the desired rotation around ( <code>xCenter, yCenter</code> ).

### Description

The function `ippiGetRotateShift` is declared in the `ippi.h` file. Use this function if you need to rotate an image about an arbitrary center (`xCenter, yCenter`) rather than the origin (0,0). The function helps compute shift values `xShift, yShift` that should be passed to [ippiRotate](#) function for the rotation around (`xCenter, yCenter`) to take place.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>xShift</code> or <code>yShift</code> pointer is NULL.

---

## AddRotateShift

*Computes shift values for rotating an image around the specified center with specified shifts.*

---

### Syntax

```
IppStatus ippiAddRotateShift(double xCenter, double yCenter,  
                             double angle, double* xShift, double* yShift)
```

### Parameters

<i>xCenter, yCenter</i>	Coordinates of the required center of rotation.
<i>angle</i>	The angle in degrees to rotate the image clockwise around the point with coordinates ( <i>xCenter, yCenter</i> ).
<i>xShift, yShift</i>	Pointers to modified shift values along horizontal and vertical axes. These new shift values should be passed to <code>ippiRotate</code> function to bring about the desired rotation around ( <i>xCenter, yCenter</i> ) and shifting.

### Description

The function `ippiAddRotateShift` is declared in the `ippi.h` file. Use this function if you need to rotate an image about an arbitrary center (*xCenter, yCenter*) rather than the origin (0,0) with required shifts. The function helps compute shift values *xShift, yShift* that should be passed to [ippiRotate](#) function to perform the rotation around (*xCenter, yCenter*) and desired shifting. The shift values should be initialized. For example, to rotate an image around a point (*xCenter, yCenter*) by the *angle* with shift values (30.3, 26.2) the following code must be written:

```
xShift = 30.3  
yShift = 26.2  
ippiAddRotateShift(xCenter, yCenter, angle, &xShift, &yShift);  
ippiRotate(angle, xShift, yShift);
```

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>xShift</i> or <i>yShift</i> pointer is NULL.

---

## GetRotateQuad

*Computes vertex coordinates of the quadrangle, to which the source ROI would be mapped by the `ippiRotate` function.*

---

### Syntax

```
IppStatus ippiGetRotateQuad(IppiRect srcRoi, double quad[4][2],  
                           double angle, double xShift, double yShift);
```

### Parameters

<i>srcRoi</i>	Source image ROI.
<i>quad</i>	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI would be mapped by <code>ippiRotate</code> function.
<i>angle</i>	The angle of rotation in degrees.
<i>xShift, yShift</i>	The shifts along horizontal and vertical axes to perform after the rotation.

### Description

The function `ippiGetRotateQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiRotate](#). It computes vertex coordinates of the quadrangle, to which the source rectangular ROI would be mapped by the `ippiRotate` function that rotates an image by *angle* degrees and shifts it by *xShift, yShift*.

The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means *x* and *y* coordinates of the vertex. Quadrangle vertices have the following meaning:

- `quad[0]` corresponds to the transformed top-left corner of the source ROI,
- `quad[1]` corresponds to the transformed top-right corner of the source ROI,
- `quad[2]` corresponds to the transformed bottom-right corner of the source ROI,
- `quad[3]` corresponds to the transformed bottom-left corner of the source ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
--------------------------	---

<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
----------------------------	---

---

## GetRotateBound

*Computes the bounding rectangle for the source ROI transformed by the `ippiRotate` function.*

---

### Syntax

```
IppStatus ippiGetRotateBound(IppiRect srcRoi, double bound[2][2],
                             double angle, double xShift, double yShift);
```

### Parameters

<i>srcRoi</i>	Source image ROI.
<i>bound</i>	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
<i>angle</i>	The angle of rotation in degrees.
<i>xShift</i> , <i>yShift</i>	The shifts along horizontal and vertical axes to perform after the rotation.

### Description

The function `ippiGetRotateBound` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiRotate](#). It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI would be mapped by the `ippiRotate` function that rotates an image by *angle* degrees and shifts it by *xShift*, *yShift*.

*bound*[0] specifies x, y coordinates of the top-left corner, *bound*[1] specifies x, y coordinates of the bottom-right corner.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

`ippStsSizeErr` Indicates an error condition if `srcRoi` has a size field with zero or negative value.

---

## RotateCenter

*Rotates an image about an arbitrary center.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippIRotateCenter_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,
    int dstStep, IppiRect dstRoi, double angle, double xCenter,
    double yCenter, int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippIRotateCenter_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi,
    double angle, double xCenter, double yCenter, int interpolation);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------

```
IppStatus ippIRotateCenter_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi,
    double angle, double xCenter, double yCenter, int interpolation);
```

Supported values for *mod*:

8u_P4R	16u_P4R	32f_P4R
--------	---------	---------

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>srcSize</i>	Size in pixels of the source image.								
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.								
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).								
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.								
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).								
<i>angle</i>	The angle of rotation in degrees.								
<i>xCenter, yCenter</i>	Center of rotation coordinates.								
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for resampling the image. Use one of the following he following values: <table data-bbox="613 867 1295 1093"> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> <tr> <td><code>IPPI_SMOOTH_EDGE</code></td><td>use edge smoothing option in addition to one of the above methods.</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation	<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing option in addition to one of the above methods.
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation								
<code>IPPI_INTER_LINEAR</code>	linear interpolation								
<code>IPPI_INTER_CUBIC</code>	cubic interpolation								
<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing option in addition to one of the above methods.								

## Description

The function `ippiRotateCenter` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function rotates the ROI of the source image by *angle* degrees (counterclockwise for positive *angle* values) around the point with coordinates *xCenter, yCenter*. The origin of the source image is implied to be in the top left corner. The result is resampled using the [interpolation method](#) specified by the *interpolation* parameter, and written to the destination image ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the intersection of the <i>srcRoi</i> and source image is less than or equal to 1.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

---

## Shear

*Performs shearing transform of the source image.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiShear_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize,
    int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep,
    IppiRect dstRoi double xShear, double yShear, double xShift,
    double yShift, int interpolation);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>32f_AC4R</code>

**Case 2: Operation on planar-order data**

```
IppStatus ippiShear_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, double xShear, double yShear,
    double xShift, double yShift, int interpolation);
```

Supported values for *mod*:

8u\_P3R          32f\_P3R

```
IppStatus ippiShear_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, double xShear, double yShear,
    double xShift, double yShift, int interpolation);
```

Supported values for *mod*:

8u\_P4R          32f\_P4R

**Parameters**

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.						
<i>srcSize</i>	Size in pixels of the source image.						
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.						
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).						
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.						
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.						
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).						
<i>xShear</i> , <i>yShear</i>	Coefficients of the shearing transform.						
<i>xShift</i> , <i>yShift</i>	Additional shift values along the horizontal and vertical axes.						
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for resampling the image. Use one of the following he following values: <table data-bbox="613 1403 1295 1516"> <tr> <td>IPPI_INTER_NN</td><td>nearest neighbor interpolation</td></tr> <tr> <td>IPPI_INTER_LINEAR</td><td>linear interpolation</td></tr> <tr> <td>IPPI_INTER_CUBIC</td><td>cubic interpolation</td></tr> </table>	IPPI_INTER_NN	nearest neighbor interpolation	IPPI_INTER_LINEAR	linear interpolation	IPPI_INTER_CUBIC	cubic interpolation
IPPI_INTER_NN	nearest neighbor interpolation						
IPPI_INTER_LINEAR	linear interpolation						
IPPI_INTER_CUBIC	cubic interpolation						



IPPI\_SMOOTH\_EDGE

use edge smoothing option in addition to one of the above methods.

### Description

The function `ippiShear` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

The shearing transform, performed by this function, maps the source image pixel coordinates  $(x,y)$  according to the following formulas:

$$x' = x + xShear * y + xShift$$

$$y' = yShear * x + y + yShift$$

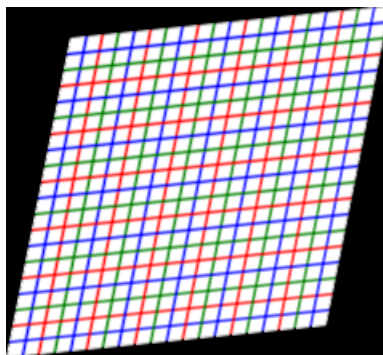
where  $x'$  and  $y'$  denote the pixel coordinates in the sheared image. The shearing transform is a special case of an affine transform, performed by the [ippiWarpAffine](#) function.

The transformed part of the image is resampled using the [interpolation method](#) specified by the *interpolation* parameter, and written to the destination image ROI.

[Figure 12-6](#) gives an example of applying the shearing transform function `ippiShear` to a sample image

**Figure 12-6 Shearing a Sample Image**

---



### Return Values

`ippStsNoErr`

Indicates no error. Any other value indicates an error or a warning.

---

<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <code>interpolation</code> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error condition if $xShear * yShear = 1$ .
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the intersection of the <code>srcRoi</code> and source image is less than or equal to 1.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <code>srcRoi</code> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

---

## GetShearQuad

*Computes vertex coordinates of the quadrangle, to which the source ROI rectangle would be mapped by the shearing transform.*

---

### Syntax

```
IppStatus ippGetShearQuad(IppiRect srcRoi, double quad[4][2],
    double xShear, double yShear, double xShift, double yShift);
```

### Parameters

<code>srcRoi</code>	Region of interest in the source image (of the <code>IppiRect</code> type).
<code>quad</code>	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI would be mapped by the shearing transform function <code>ippiShear</code> .

<code>xShear, yShear</code>	Shearing transform coefficients.
<code>xShift, yShift</code>	Additional shift values along the horizontal and vertical axes.

## Description

The function `ippiGetShearQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiShear](#). It computes vertex coordinates of the quadrangle, to which the source rectangular ROI would be mapped by the shearing transform function `ippiShear` using coefficients `xShear, yShear` and shift values `xShift, yShift`. The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

<code>quad[0]</code>	corresponds to the transformed top-left corner of the source ROI,
<code>quad[1]</code>	corresponds to the transformed top-right corner of the source ROI,
<code>quad[2]</code>	corresponds to the transformed bottom-right corner of the source ROI,
<code>quad[3]</code>	corresponds to the transformed bottom-left corner of the source ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsCoeffErr</code>	Indicates an error condition if $xShear * yShear = 1$ .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoi</code> has a size field with zero or negative value.

---

## GetShearBound

*Computes the bounding rectangle for the source ROI transformed by the `ippiShear` function.*

---

## Syntax

```
IppStatus ippiGetShearBound(IppiRect srcRoi, double bound[2][2],
    double xShear, double yShear, double xShift, double yShift);
```

### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>bound</i>	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
<i>xShear</i> , <i>yShear</i>	Shearing transform coefficients.
<i>xShift</i> , <i>yShift</i>	Additional shift values along the horizontal and vertical axes.

### Description

The function `ippiGetShearBound` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiShear](#). It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI would be mapped by the shearing transform function `ippiShear` using coefficients *xShear*, *yShear* and shift values *xShift*, *yShift*.

*bound*[0] specifies x, y coordinates of the top-left corner, *bound*[1] specifies x, y coordinates of the bottom-right corner.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsCoeffErr</code>	Indicates an error condition if $xShear * yShear = 1$ .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.

---

## WarpAffine

*Performs general affine transform of the source image.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpAffine_<mod>(const Ipp<datatype>* pSrc,  
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,  
    int dstStep, IppiRect dstRoi, const double coeffs[2][3],  
    int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

### Case 2: Operation on planar-order data

```
IppStatus ippiWarpAffine_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[3], int dstStep, IppiRect dstRoi,
    const double coeffs[2][3], int interpolation);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------

```
IppStatus ippiWarpAffine_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi,
    Ipp<datatype>* const pDst[4], int dstStep, IppiRect dstRoi,
    const double coeffs[2][3], int interpolation);
```

Supported values for *mod*:

8u_P4R	16u_P4R	32f_P4R
--------	---------	---------

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).
<i>coeffs</i>	The affine transform coefficients.

<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for resampling the image. Use one of the following he following values:	
	IPPI_INTER_NN	nearest neighbor interpolation
	IPPI_INTER_LINEAR	linear interpolation
	IPPI_INTER_CUBIC	cubic interpolation
	IPPI_SMOOTH_EDGE	use edge smoothing option in addition to one of the above methods.

## Description

The function `ippiWarpAffine` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This affine warp function transforms the source image pixel coordinates  $(x,y)$  according to the following formulas:

$$\begin{aligned}x' &= c_{00}*x + c_{01}*y + c_{02} \\ y' &= c_{10}*x + c_{11}*y + c_{12}\end{aligned}$$

where  $x'$  and  $y'$  denote the pixel coordinates in the transformed image, and  $c_{ij}$  are the affine transform coefficients passed in the array `coeffs`.

The affine warping is a general linear transform which incorporates such elementary transformations as scaling, rotation, translation, stretching, and shearing. It always transforms parallel lines into parallel lines and preserves equal distances between points on a line.

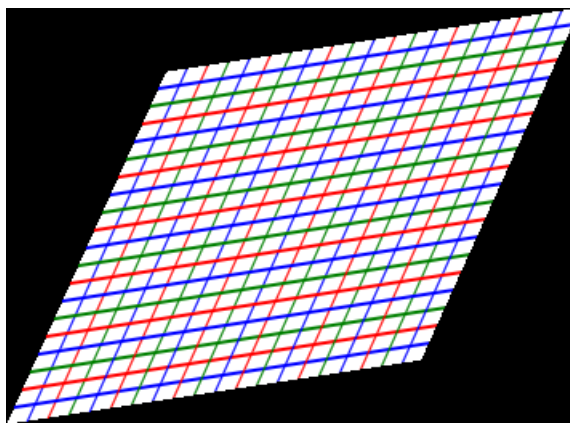
The transformed part of the image is resampled using the [interpolation method](#) specified by the *interpolation* parameter, and written to the destination image ROI.

[Figure 12-7](#) gives an example of applying the affine warping function `ippiWarpAffine` to a sample image.

To compute the affine transform parameters, use the [ippiGetAffineQuad](#), [ippiGetAffineBound](#), and [ippiGetAffineTransform](#) functions described later in this chapter.

**Figure 12-7 Affine Transform of an Image**

---



## Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippiStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippiStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippiStsInterpolationErr</code>	Indicates an error condition if <code>interpolation</code> has an illegal value.
<code>ippiStsCoeffErr</code>	Indicates an error condition if $c_{00} * c_{11} - c_{01} * c_{10} = 0$ .
<code>ippiStsRectErr</code>	Indicates an error condition if width or height of the intersection of the <code>srcRoi</code> and source image is less than or equal to 1.

---

<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

---

## WarpAffineBack

*Performs an inverse affine transform of an image.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippWarpAffineBack_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,
    int dstStep, IppiRect dstRoi, const double coeffs[2][3],
    int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippWarpAffineBack_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, const double coeffs[2][3],
    int interpolation);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------

```
IppStatus ippWarpAffineBack_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, const double coeffs[2][3],
    int interpolation);
```



Supported values for *mod*:

8u\_P4R          16u\_P4R          32f\_P4R

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.						
<i>srcSize</i>	Size in pixels of the source image.						
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.						
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).						
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.						
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.						
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).						
<i>coeffs</i>	The affine transform coefficients.						
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for resampling the image. Use one of the following he following values: <table data-bbox="682 930 1364 1050"> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation						
<code>IPPI_INTER_LINEAR</code>	linear interpolation						
<code>IPPI_INTER_CUBIC</code>	cubic interpolation						

## Description

The function `ippiWarpAffineBack` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function performs the inverse transform to that defined by [ippiWarpAffine](#) function. Pixel coordinates  $x'$  and  $y'$  in the transformed image are obtained from the following equations:

$$c_{00} * x' + c_{01} * y' + c_{02} = x$$

$$c_{10} * x' + c_{11} * y' + c_{12} = y$$

where  $x$  and  $y$  denote the pixel coordinates in the source image, and coefficients  $c_{ij}$  are given in the array *coeffs*. Thus, you do not need to invert transform coefficients in your application program before calling `ippiWarpAffineBack`.

Note that inverse transform functions handle source and destination ROI in a different way than other geometric transform functions. Their implementation include the following logic:

- backward transform is applied to coordinates of each pixel in the destination ROI, which gives as a result coordinates of some pixel in the source image
- if the obtained source pixel is inside source ROI, the corresponding pixel in destination ROI is modified accordingly; otherwise no change is made.

The above algorithm can be represented in pseudocode as follows:

```
for (j = dstRoi.y; j < dstRoi.y + dstRoi.height; j++) {
    for (i = dstRoi.x; i < dstRoi.x + dstRoi.width; i++) {
        sx = TransformX(i, j);
        sy = TransformY(i, j);
        if (sx >= srcRoi.x && sx < srcRoi.x + srcRoi.width && sy >=
srcRoi.y && sy < srcRoi.y + srcRoi.height) {
            dst[i, j] = Interpolate(sx, sy);}
    }
}
```

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error condition if $c_{00} * c_{11} - c_{01} * c_{10} = 0$ .
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.

ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.
--------------------------	--

---

## WarpAffineQuad

*Performs image affine warping that transforms the given source quadrangle to the specified destination quadrangle.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
ippStatus ippWarpAffineQuad_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, const double
    srcQuad[4][2], Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi,
    const double dstQuad[4][2], int interpolation);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R
8u_AC4R	16u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
ippStatus ippWarpAffineQuad_<mod>(const Ipp<datatype>*
    const pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcRoi,
    const double srcQuad[4][2], Ipp<datatype>* const pDst[3],
    int dstStep, IppiRect dstRoi, const double dstQuad[4][2],
    int interpolation);
```

Supported values for *mod*:

8u_P3R	16u_P3R	32f_P3R
--------	---------	---------

```
ippStatus ippWarpAffineQuad_<mod>(const Ipp<datatype>*
    const pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcRoi,
    const double srcQuad[4][2], Ipp<datatype>* const pDst[4],
    int dstStep, IppiRect dstRoi, const double dstQuad[4][2],
    int interpolation);
```

Supported values for *mod*:

8u\_P4R      16u\_P4R      32f\_P4R

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>srcSize</i>	Size in pixels of the source image.								
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.								
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).								
<i>srcQuad</i>	A given quadrangle in the source image.								
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.								
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).								
<i>dstQuad</i>	A given quadrangle in the destination image.								
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for resampling the image. Use one of the following he following values: <table data-bbox="617 982 1299 1212"> <tr> <td>IPPI_INTER_NN</td><td>nearest neighbor interpolation</td></tr> <tr> <td>IPPI_INTER_LINEAR</td><td>linear interpolation</td></tr> <tr> <td>IPPI_INTER_CUBIC</td><td>cubic interpolation</td></tr> <tr> <td>IPPI_SMOOTH_EDGE</td><td>use edge smoothing option in addition to one of the above methods.</td></tr> </table>	IPPI_INTER_NN	nearest neighbor interpolation	IPPI_INTER_LINEAR	linear interpolation	IPPI_INTER_CUBIC	cubic interpolation	IPPI_SMOOTH_EDGE	use edge smoothing option in addition to one of the above methods.
IPPI_INTER_NN	nearest neighbor interpolation								
IPPI_INTER_LINEAR	linear interpolation								
IPPI_INTER_CUBIC	cubic interpolation								
IPPI_SMOOTH_EDGE	use edge smoothing option in addition to one of the above methods.								

## Description

The function `ippiWarpAffineQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function applies the affine transform to an arbitrary quadrangle *srcQuad* in the source image *pSrc*. The operations take place only inside the intersection of the source image ROI *srcRoi* and the source quadrangle. The function `ippiWarpAffineQuad` uses the same formulas for pixel mapping as in the case of the [ippiWarpAffine](#) function. Transform coefficients are computed

internally, based on the mapping of the source quadrangle to the quadrangle *dstQuad* specified in the destination image *pDst*. The *dstQuad* should have a non-empty intersection with the destination image ROI *dstRoi*. The function computes the coordinates of the 4th vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the ones specified in *dstQuad*, the function returns the warning message and continues operation with the computed values.

The first dimension [4] of the array specifying the quadrangle *srcQuad*[4][2] or *dstQuad*[4][2] is equal to the number of vertices, and the second dimension [2] holds x and y coordinates of the vertex.

Edge smoothing interpolation is applicable only if the source quadrangle lies in the source image ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsQuadErr</code>	Indicates an error condition if <i>srcQuad</i> or <i>dstQuad</i> degenerates into triangle, line, or point, or if <i>dstQuad</i> has wrong vertex coordinates.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the <i>srcRoi</i> has no intersection with the <i>srcQuad</i> , or <i>dstRoi</i> has no intersection with the <i>dstQuad</i> .
<code>ippStsAffineQuadChanged</code>	Indicates a warning that coordinates of the 4th vertex of the destination quadrangle <i>dstQuad</i> are corrected by the function.

## GetAffineQuad

*Computes vertex coordinates of the quadrangle, to which the source ROI rectangle would be mapped by the affine transform.*

### Syntax

```
IppStatus ippiGetAffineQuad(IppiRect srcRoi, double quad[4][2],
    const double coeffs[2][3]);
```

### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>quad</i>	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI would be mapped by the affine transform function <code>ippiWarpAffine</code> .
<i>coeffs</i>	The given affine transform coefficients.

### Description

The function `ippiGetAffineQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiWarpAffine](#). It computes vertex coordinates of the quadrangle, to which the source rectangular ROI would be mapped by the affine transform function `ippiWarpAffine` using the given coefficients *coeffs*.

The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

- `quad[0]` corresponds to the transformed top-left corner of the source ROI,
- `quad[1]` corresponds to the transformed top-right corner of the source ROI,
- `quad[2]` corresponds to the transformed bottom-right corner of the source ROI,
- `quad[3]` corresponds to the transformed bottom-left corner of the source ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsCoeffErr</code>	Indicates an error condition if $c_{00} * c_{11} - c_{01} * c_{10} = 0$ .

<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoi</code> has a size field with zero or negative value.
----------------------------	---

---

## GetAffineBound

*Computes the bounding rectangle for the source ROI transformed by the `ippiWarpAffine` function.*

---

### Syntax

```
IppStatus ippiGetAffineBound(IppiRect srcRoi, double bound[2][2],  
                             const double coeffs[2][3]);
```

### Parameters

<code>srcRoi</code>	Region of interest in the source image (of the <code>IppiRect</code> type).
<code>bound</code>	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
<code>coeffs</code>	The given affine transform coefficients.

### Description

The function `ippiGetAffineBound` is declared in the `ippi.h` file. This function is used as a support function for [ippiWarpAffine](#). It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI would be mapped by the affine transform function `ippiWarpAffine` using coefficients *coeffs*.

*bound*[0] specifies x, y coordinates of the top-left corner, *bound*[1] specifies x, y coordinates of the bottom-right corner.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsCoeffErr</code>	Indicates an error condition if $c_{00} * c_{11} - c_{01} * c_{10} = 0$ .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoi</code> has a size field with zero or negative value.

---

## GetAffineTransform

*Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates*

---

### Syntax

```
IppStatus ippiGetAffineTransform(IppiRect srcRoi, const double quad[4][2],  
                                double coeffs[2][3]);
```

### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>quad</i>	Vertex coordinates of the quadrangle, to which the source ROI would be mapped by the affine transform function <code>ippiWarpAffine</code> .
<i>coeffs</i>	Output array. Contains the target affine transform coefficients.

### Description

The function `ippiGetAffineTransform` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiWarpAffine](#). It computes the coefficients *coeffs* of the affine transform that should be used by the function `ippiWarpAffine` to map the source rectangular ROI to the quadrangle with the specified vertex coordinates *quad*.

The first dimension [4] of the array *quad*[4][2] is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

- quad*[0] corresponds to the transformed top-left corner of the source ROI,
- quad*[1] corresponds to the transformed top-right corner of the source ROI,
- quad*[2] corresponds to the transformed bottom-right corner of the source ROI,
- quad*[3] corresponds to the transformed bottom-left corner of the source ROI.

The function computes the coordinates of the 4th vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the ones specified in *quad*, the function returns the warning message and continues operation with the computed values.



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or warning.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the <i>srcRoi</i> is less than or equal to 1.
<code>ippStsCoeffErr</code>	Indicates an error condition if $c_{00} * c_{11} - c_{01} * c_{10} = 0$ .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
<code>ippStsAffineQuadChanged</code>	Indicates a warning that coordinates of the 4th vertex of the specified quadrangle <i>quad</i> are not correct.

---

## WarpPerspective

*Performs perspective warping of the source image using the given transform coefficients.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippkWarpPerspective_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,
    int dstStep, IppiRect dstRoi, const double coeffs[3][3],
    int interpolation);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>32f_AC4R</code>

#### Case 2: Operation on planar-order data

```
IppStatus ippkWarpPerspective_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, const double coeffs[3][3],
    int interpolation);
```

Supported values for *mod*:

8u\_P3R            32f\_P3R

```
IppStatus ippiWarpPerspective_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, const double coeffs[3][3],
    int interpolation);
```

Supported values for *mod*:

8u\_P4R            32f\_P4R

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>srcSize</i>	Size in pixels of the source image.								
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.								
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).								
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.								
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).								
<i>coeffs</i>	The perspective transform coefficients.								
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for resampling the image. Use one of the following he following values: <table data-bbox="617 1175 1299 1397"> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> <tr> <td><code>IPPI_SMOOTH_EDGE</code></td><td>use edge smoothing option in addition to one of the above methods.</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation	<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing option in addition to one of the above methods.
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation								
<code>IPPI_INTER_LINEAR</code>	linear interpolation								
<code>IPPI_INTER_CUBIC</code>	cubic interpolation								
<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing option in addition to one of the above methods.								

## Description

The function `ippiWarpPerspective` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This perspective warp function transforms the source image pixel coordinates  $(x,y)$  according to the following formulas:

$$x' = (c_{00}*x + c_{01}*y + c_{02}) / (c_{20}*x + c_{21}*y + c_{22})$$
$$y' = (c_{10}*x + c_{11}*y + c_{12}) / (c_{20}*x + c_{21}*y + c_{22})$$

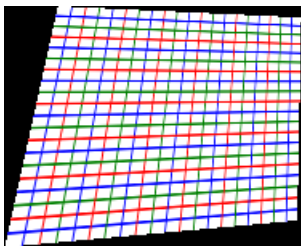
where  $x'$  and  $y'$  denote the pixel coordinates in the transformed image, and  $c_{ij}$  are the perspective transform coefficients passed in the array `coeffs`.

The transformed part of the image is resampled using the [interpolation method](#) specified by the `interpolation` parameter, and written to the destination image ROI.

[Figure 12-8](#) gives an example of applying the perspective transform function `ippiWarpPerspective` to a sample image.

**Figure 12-8**    **Perspective Transform of an Image**

---



To estimate how the source image ROI will be transformed by the `ippiWarpPerspective` function, use functions [ippiWarpPerspectiveQuad](#) and [ippiGetPerspectiveBound](#) described in the sections that follow. To calculate coefficients of the perspective transform which maps source ROI to a given quadrangle, use [ippiGetPerspectiveTransform](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.

<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the intersection of the <i>srcRoi</i> and source image is less than or equal to 1.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

---

## WarpPerspectiveBack

*Performs an inverse perspective warping of the source image.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippkWarpPerspectiveBack_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,
    int dstStep, IppiRect dstRoi, const double coeffs[3][3],
    int interpolation);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>32f_AC4R</code>

### Case 2: Operation on planar-order data

```
IppStatus ippiWarpPerspectiveBack_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, const double coeffs[3][3],
    int interpolation);
```

Supported values for *mod*:

8u\_P3R            32f\_P3R

```
IppStatus ippiWarpPerspectiveBack_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, const double coeffs[3][3],
    int interpolation);
```

Supported values for *mod*:

8u\_P4R            32f\_P4R

### Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image (of the <code>IppiRect</code> type).
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>srcRoi</i>	Region of interest in the source image.
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).
<i>coeffs</i>	The perspective transform coefficients.
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for resampling the image. Use one of the following he following values:

<code>IPPI_INTER_NN</code>	nearest neighbor interpolation
<code>IPPI_INTER_LINEAR</code>	linear interpolation
<code>IPPI_INTER_CUBIC</code>	cubic interpolation

## Description

The function `ippiWarpPerspectiveBack` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function performs the inverse transform to that defined by [ippiWarpPerspective](#) function. Pixel coordinates  $x'$  and  $y'$  in the transformed image are obtained from the following equations

$$\begin{aligned}(c_{00}*x' + c_{01}*y' + c_{02})/(c_{20}*x' + c_{21}*y' + c_{22}) &= x \\ (c_{10}*x' + c_{11}*y' + c_{12})/(c_{20}*x' + c_{21}*y' + c_{22}) &= y\end{aligned}$$

where  $x$  and  $y$  denote the pixel coordinates in the source image, and coefficients  $c_{ij}$  are given in the array `coeffs`. Thus, you don't need to invert transform coefficients in your application program before calling `ippiWarpPerspectiveBack`.

Note that inverse transform functions handle source and destination ROI in a different way than other geometric transform functions. See implementation details in the description of [ippiWarpAffineBack](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <code>interpolation</code> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <code>srcRoi</code> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

---

## WarpPerspectiveQuad

*Performs perspective warping of the given source quadrangle to the specified destination quadrangle.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpPerspectiveQuad_<mod>(const Ipp<datatype>* pSrc,  
    IppiSize srcSize, int srcStep, IppiRect srcRoi, const double  
    srcQuad[4][2], Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi,  
    const double dstQuad[4][2], int interpolation);
```

Supported values for *mod*:

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippiWarpPerspectiveQuad_<mod>(const Ipp<datatype>*  
    const pSrc[3], IppiSize srcSize, int srcStep, IppiRect srcRoi,  
    const double srcQuad[4][2], Ipp<datatype>* const pDst[3],  
    int dstStep, IppiRect dstRoi, const double dstQuad[4][2],  
    int interpolation);
```

Supported values for *mod*:

8u_P3R	32f_P3R
--------	---------

```
IppStatus ippiWarpPerspectiveQuad_<mod>(const Ipp<datatype>*  
    const pSrc[4], IppiSize srcSize, int srcStep, IppiRect srcRoi,  
    const double srcQuad[4][2], Ipp<datatype>* const pDst[4],  
    int dstStep, IppiRect dstRoi, const double dstQuad[4][2],  
    int interpolation);
```

Supported values for *mod*:

8u_P4R	32f_P4R
--------	---------

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.						
<i>srcSize</i>	Size in pixels of the source image.						
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.						
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).						
<i>srcQuad</i>	A given quadrangle in the source image.						
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.						
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.						
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).						
<i>dstQuad</i>	A given quadrangle in the destination image.						
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for resampling the image. Use one of the following values: <table data-bbox="617 871 1299 983"> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation						
<code>IPPI_INTER_LINEAR</code>	linear interpolation						
<code>IPPI_INTER_CUBIC</code>	cubic interpolation						

## Description

The function `ippiWarpPerspectiveQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function applies a perspective transform to an arbitrary quadrangle *srcQuad* in the source image *pSrc*. The operations take place only in the intersection of the source image ROI *srcRoi* and the source quadrangle. The function `ippiWarpPerspectiveQuad` uses the same formulas for pixel mapping as in the case of the [ippiWarpPerspective](#) function. Transform coefficients are computed internally, based on the mapping of the source quadrangle to the quadrangle *dstQuad* specified in the destination image *pDst*. The *dstQuad* should have a non-empty intersection with the destination image ROI *dstRoi*.

The first dimension [4] of the array specifying the quadrangle *srcQuad*[4][2] or *dstQuad*[4][2] is equal to the number of vertices, and the second dimension [2] holds x and y coordinates of the vertex.



Edge smoothing interpolation is applicable only if the source quadrangle lies in the source image ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsQuadErr</code>	Indicates an error condition if <i>srcQuad</i> or <i>dstQuad</i> degenerates into triangle.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the <i>srcRoi</i> has no intersection with the <i>srcQuad</i> , or <i>dstRoi</i> has no intersection with the <i>dstQuad</i> .

---

## GetPerspectiveQuad

*Computes vertex coordinates of the quadrangle, to which the source ROI rectangle would be mapped by the perspective transform.*

---

### Syntax

```
IppStatus ippiGetPerspectiveQuad(IppiRect srcRoi, double quad[4][2],
    const double coeffs[3][3]);
```

### Parameters

*srcRoi*                      Region of interest in the source image (of the `IppiRect` type).

<i>quad</i>	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI would be mapped by the perspective transform function <code>ippiWarpPerspective</code> .
<i>coeffs</i>	The given perspective transform coefficients.

## Description

The function `ippiGetPerspectiveQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiWarpPerspective](#). It computes vertex coordinates of the quadrangle, to which the source rectangular ROI would be mapped by the perspective transform function `ippiWarpPerspective` using the given coefficients *coeffs*.

The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

- `quad[0]` corresponds to the transformed top-left corner of the source ROI,
- `quad[1]` corresponds to the transformed top-right corner of the source ROI,
- `quad[2]` corresponds to the transformed bottom-right corner of the source ROI,
- `quad[3]` corresponds to the transformed bottom-left corner of the source ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.

---

## GetPerspectiveBound

*Computes the bounding rectangle for the source ROI transformed by the `ippiWarpPerspective` function.*

---

## Syntax

```
IppStatus ippiGetPerspectiveBound(IppiRect srcRoi, double bound[2][2],
    const double coeffs[3][3]);
```

### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>bound</i>	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
<i>coeffs</i>	The given perspective transform coefficients.

### Description

The function `ippiGetPerspectiveBound` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiWarpPerspective](#). It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI would be mapped by the perspective transform function `ippiWarpPerspective` using the given coefficients *coeffs*.

*bound*[0] specifies x, y coordinates of the top-left corner, *bound*[1] specifies x, y coordinates of the bottom-right corner.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.

---

## GetPerspectiveTransform

*Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.*

---

### Syntax

```
IppStatus ippiGetPerspectiveTransform(IppiRect srcRoi,  
    const double quad[4][2], double coeffs[3][3]);
```

## Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>quad</i>	Vertex coordinates of the quadrangle, to which the source ROI would be mapped by the perspective transform function <code>ippiWarpPerspective</code> .
<i>coeffs</i>	Output array. Contains the target perspective transform coefficients.

## Description

The function `ippiGetPerspectiveTransform` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiWarpPerspective](#). It computes the coefficients *coeffs* that should be used by the function `ippiWarpPerspective` to map the source rectangular ROI to the quadrangle with the given vertex coordinates *quad*.

The first dimension [4] of the array *quad*[4] [2] is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

- quad*[0] corresponds to the transformed top-left corner of the source ROI,
- quad*[1] corresponds to the transformed top-right corner of the source ROI,
- quad*[2] corresponds to the transformed bottom-right corner of the source ROI,
- quad*[3] corresponds to the transformed bottom-left corner of the source ROI.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the <i>srcRoi</i> is less than or equal to 1.

---

## WarpBilinear

*Performs bilinear warping of the source image using the specified transform coefficients.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatusippiWarpBilinear_<mod>(constIpp<datatype>*pSrc,IppiSizeSize,
    int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep,
    IppiRect dstRoi, const double coeffs[2][4], int interpolation);
```

Supported values for *mod*:

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatusippiWarpBilinear_<mod>(constIpp<datatype>*constpSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>*const
    pDst[3], int dstStep, IppiRect dstRoi, const double coeffs[2][4],
    int interpolation);
```

Supported values for *mod*:

8u_P3R	32f_P3R
--------	---------

```
IppStatusippiWarpBilinear_<mod>(constIpp<datatype>*constpSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>*const
    pDst[4], int dstStep, IppiRect dstRoi, const double coeffs[2][4],
    int interpolation);
```

Supported values for *mod*:

8u_P4R	32f_P4R
--------	---------

### Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
<i>srcSize</i>	Size in pixels of the source image.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.								
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).								
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.								
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).								
<i>coeffs</i>	The bilinear transform coefficients.								
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for resampling the image. Use one of the following values: <table data-bbox="613 645 1295 865"> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> <tr> <td><code>IPPI_SMOOTH_EDGE</code></td><td>use edge smoothing option in addition to one of the above methods.</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation	<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing option in addition to one of the above methods.
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation								
<code>IPPI_INTER_LINEAR</code>	linear interpolation								
<code>IPPI_INTER_CUBIC</code>	cubic interpolation								
<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing option in addition to one of the above methods.								

## Description

The function `ippiWarpBilinear` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This bilinear warp function transforms the source image pixel coordinates  $(x,y)$  according to the following formulas:

$$x' = c_{00} * xy + c_{01} * x + c_{02} * y + c_{03}$$

$$y' = c_{10} * xy + c_{11} * x + c_{12} * y + c_{13}$$

where  $x'$  and  $y'$  denote the pixel coordinates in the transformed image, and  $c_{ij}$  are the bilinear transform coefficients passed in the array *coeffs*.

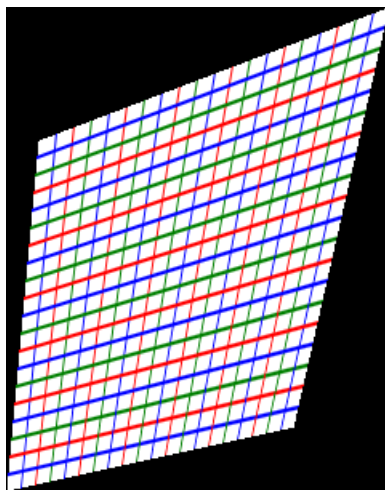
The bilinear transform preserves equal distances between points on a line.

The transformed part of the source image is resampled using the [interpolation method](#) specified by the *interpolation* parameter, and written to the destination image ROI.

[Figure 12-9](#) gives an example of applying the bilinear warping function `ippiWarpBilinear` to a sample image.

**Figure 12-9 Bilinear Transform of an Image**

---



To estimate how the source image ROI will be transformed by the `ippiWarpBilinear` function, use functions [ippiWarpBilinearQuad](#) and [ippiGetBilinearBound](#) described in the sections that follow. To calculate coefficients of the bilinear transform which maps source ROI to a given quadrangle, use [ippiGetBilinearTransform](#) function.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrc</code> or <code>pDst</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <code>interpolation</code> has an illegal value.

<code>ippStsRectErr</code>	Indicates an error condition if width or height of the intersection of the <i>srcRoi</i> and source image is less than or equal to 1.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

---

## WarpBilinearBack

*Performs an inverse bilinear warping of the source image.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippkWarpBilinearBack_<mod>(const Ipp<datatype>* pSrc,
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst,
    int dstStep, IppiRect dstRoi, const double coeffs[2][4], int interpolation);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>32f_C3R</code>
<code>8u_C4R</code>	<code>32f_C4R</code>
<code>8u_AC4R</code>	<code>32f_AC4R</code>

#### Case 2: Operation on planar-order data

```
IppStatus ippkWarpBilinearBack_<mod>(const Ipp<datatype>* const pSrc[3],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[3], int dstStep, IppiRect dstRoi, const double coeffs[2][4],
    int interpolation);
```

Supported values for *mod*:

<code>8u_P3R</code>	<code>32f_P3R</code>
---------------------	----------------------



```
IppStatus ippiWarpBilinearBack_<mod>(const Ipp<datatype>* const pSrc[4],
    IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* const
    pDst[4], int dstStep, IppiRect dstRoi, const double coeffs[2][4],
    int interpolation);
```

Supported values for *mod*:

```
8u_P4R      32f_P4R
```

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.						
<i>srcSize</i>	Size in pixels of the source image.						
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.						
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).						
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.						
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.						
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).						
<i>coeffs</i>	The bilinear transform coefficients.						
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for resampling the image. Use one of the following values: <table data-bbox="682 1076 1364 1190"> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation						
<code>IPPI_INTER_LINEAR</code>	linear interpolation						
<code>IPPI_INTER_CUBIC</code>	cubic interpolation						

## Description

The function `ippiWarpBilinearBack` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function performs the inverse transform to that defined by [ippiWarpBilinear](#) function. Pixel coordinates  $x'$  and  $y'$  in the transformed image are obtained from the following equations

$$c_{00} * x' y' + c_{01} * x' + c_{02} * y' + c_{03} = x$$

$$c_{10} * x' y' + c_{11} * x' + c_{12} * y' + c_{13} = y$$

where  $x$  and  $y$  denote the pixel coordinates in the source image, and coefficients  $c_{ij}$  are given in the array *coeffs*. Thus, you don't need to invert transform coefficients in your application program before calling `ippiWarpBilinearBack`.

Note that inverse transform functions handle source and destination ROI in a different way than other geometric transform functions. See implementation details in the description of [ippiWarpAffineBack](#) function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

---

## WarpBilinearQuad

*Performs bilinear warping of the source image that transforms the given source quadrangle to the specified destination quadrangle.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>* pSrc,  
    IppiSize srcSize, int srcStep, IppiRect srcRoi,  
    const double srcQuad[4][2], Ipp<datatype>* pDst, int dstStep,  
    IppiRect dstRoi, const double dstQuad[4][2], int interpolation);
```

Supported values for *mod*:

8u_C1R	32f_C1R
8u_C3R	32f_C3R
8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

#### Case 2: Operation on planar-order data

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>* const pSrc[3],  
    IppiSize srcSize, int srcStep, IppiRect srcRoi,  
    const double srcQuad[4][2], Ipp<datatype>* const pDst[3],  
    int dstStep, IppiRect dstRoi, const double dstQuad[4][2],  
    int interpolation);
```

Supported values for *mod*:

8u_P3R	32f_P3R
--------	---------

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>* const pSrc[4],  
    IppiSize srcSize, int srcStep, IppiRect srcRoi,  
    const double srcQuad[4][2], Ipp<datatype>* const pDst[4],  
    int dstStep, IppiRect dstRoi, const double dstQuad[4][2],  
    int interpolation);
```

Supported values for *mod*:

8u_P4R	32f_P4R
--------	---------

## Parameters

<i>pSrc</i>	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>srcSize</i>	Size in pixels of the source image.								
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.								
<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).								
<i>srcQuad</i>	A given quadrangle in the source image.								
<i>pDst</i>	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.								
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.								
<i>dstRoi</i>	Region of interest in the destination image (of the <code>IppiRect</code> type).								
<i>dstQuad</i>	A given quadrangle in the destination image.								
<i>interpolation</i>	The type of <a href="#">interpolation</a> to perform for resampling the image. Use one of the following values: <table data-bbox="617 871 1299 1093"> <tr> <td><code>IPPI_INTER_NN</code></td><td>nearest neighbor interpolation</td></tr> <tr> <td><code>IPPI_INTER_LINEAR</code></td><td>linear interpolation</td></tr> <tr> <td><code>IPPI_INTER_CUBIC</code></td><td>cubic interpolation</td></tr> <tr> <td><code>IPPI_SMOOTH_EDGE</code></td><td>use edge smoothing option in addition to one of the above methods.</td></tr> </table>	<code>IPPI_INTER_NN</code>	nearest neighbor interpolation	<code>IPPI_INTER_LINEAR</code>	linear interpolation	<code>IPPI_INTER_CUBIC</code>	cubic interpolation	<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing option in addition to one of the above methods.
<code>IPPI_INTER_NN</code>	nearest neighbor interpolation								
<code>IPPI_INTER_LINEAR</code>	linear interpolation								
<code>IPPI_INTER_CUBIC</code>	cubic interpolation								
<code>IPPI_SMOOTH_EDGE</code>	use edge smoothing option in addition to one of the above methods.								

## Description

The function `ippiWarpBilinearQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function applies a bilinear transform to an arbitrary quadrangle *srcQuad* in the source image *pSrc*. The operations take place only in the intersection of the source image ROI *srcRoi* and the source quadrangle *srcQuad*. The function `ippiWarpBilinearQuad` uses the same formulas for pixel mapping as in the case of the [ippiWarpBilinear](#) function. Transform coefficients are computed internally, based on the mapping of the source quadrangle to the quadrangle *dstQuad* specified in the destination image *pDst*. The *dstQuad* should have a non-empty intersection with the destination image ROI *dstRoi*.

The first dimension [4] of the array specifying the quadrangle *srcQuad*[4][2] or *dstQuad*[4][2] is equal to the number of vertices, and the second dimension [2] holds x and y coordinates of the vertex.

Edge smoothing interpolation is applicable only if the source quadrangle lies in the source image ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrc</i> or <i>pDst</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if any image dimension has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.
<code>ippStsInterpolationErr</code>	Indicates an error condition if <i>interpolation</i> has an illegal value.
<code>ippStsQuadErr</code>	Indicates an error condition if <i>srcQuad</i> or <i>dstQuad</i> degenerates into triangle.
<code>ippStsWrongIntersectROIErr</code>	Indicates an error condition if <i>srcRoi</i> has no intersection with the source image.
<code>ippStsWrongIntersectQuad</code>	Indicates a warning that no operation is performed if the <i>srcRoi</i> has no intersection with the <i>srcQuad</i> , or <i>dstRoi</i> has no intersection with the <i>dstQuad</i> .

---

## GetBilinearQuad

*Computes the vertex coordinates of the quadrangle, to which the source rectangular ROI would be mapped by the bilinear transform.*

---

### Syntax

```
IppStatus ippiGetBilinearQuad(IppiRect srcRoi, double quad[4][2],  
    const double coeffs[2][4]);
```

### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>quad</i>	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI would be mapped by the bilinear transform function <code>ippiWarpBilinear</code> .
<i>coeffs</i>	The given bilinear transform coefficients.

### Description

The function `ippiGetBilinearQuad` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiWarpBilinear](#). It computes vertex coordinates of the quadrangle, to which the source rectangular ROI would be mapped by the bilinear transform function `ippiWarpBilinear` using coefficients *coeffs*.

The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

- `quad[0]` corresponds to the transformed top-left corner of the source ROI,
- `quad[1]` corresponds to the transformed top-right corner of the source ROI,
- `quad[2]` corresponds to the transformed bottom-right corner of the source ROI,
- `quad[3]` corresponds to the transformed bottom-left corner of the source ROI.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
--------------------------	---

<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.

---

## GetBilinearBound

*Computes the bounding rectangle for the source ROI transformed by the `ippiWarpBilinear` function.*

---

### Syntax

```
IppStatus ippiGetBilinearBound(IppiRect srcRoi, double bound[2][2],  
    const double coeffs[2][4]);
```

### Parameters

<i>srcRoi</i>	Region of interest in the source image (of the <code>IppiRect</code> type).
<i>bound</i>	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
<i>coeffs</i>	The given bilinear transform coefficients.

### Description

The function `ippiGetBilinearBound` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiWarpBilinear](#). It computes vertex coordinates of the smallest bounding rectangle for the quadrangle *quad*, to which the source ROI would be mapped by the bilinear transform function `ippiWarpBilinear` using coefficients *coeffs*.

*bound*[0] specifies x, y coordinates of the top-left corner, *bound*[1] specifies x, y coordinates of the bottom-right corner.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.

`ippStsCoeffErr` Indicates an error condition if coefficient values are invalid.

---

## GetBilinearTransform

*Computes bilinear transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.*

---

### Syntax

```
IppStatus ippiGetBilinearTransform(IppiRect srcRoi, const double quad[4][2],
                                   double coeffs[2][4]);
```

### Parameters

<code>srcRoi</code>	Region of interest in the source image (of the <code>IppiRect</code> type).
<code>quad</code>	Vertex coordinates of the quadrangle, to which the source ROI would be mapped by the bilinear transform function <code>ippiWarpBilinear</code> .
<code>coeffs</code>	Output array. Contains the target bilinear transform coefficients.

### Description

The function `ippiGetBilinearTransform` is declared in the `ippi.h` file. It operates with ROI (see [ROI Processing in Geometric Transforms](#)).

This function is used as a support function for [ippiWarpBilinear](#). It computes the coefficients `coeffs` of the bilinear transform that maps the source rectangular ROI to the quadrangle with the specified vertex coordinates `quad`.

The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

- `quad[0]` corresponds to the transformed top-left corner of the source ROI,
- `quad[1]` corresponds to the transformed top-right corner of the source ROI,
- `quad[2]` corresponds to the transformed bottom-right corner of the source ROI,
- `quad[3]` corresponds to the transformed bottom-left corner of the source ROI.



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoi</i> has a size field with zero or negative value.
<code>ippStsCoeffErr</code>	Indicates an error condition if coefficient values are invalid.
<code>ippStsRectErr</code>	Indicates an error condition if width or height of the <i>srcRoi</i> is less than or equal to 1.

# Wavelet Transforms

# 13

This chapter describes two-dimensional Discrete Wavelet Transform (DWT) functions implemented in the Intel IPP for image processing.

In many applications, the multiresolution analysis by discrete wavelet transforms is a better alternative to windowing and discrete Fourier analysis techniques. On the one hand, the forward two-dimensional wavelet transform may be considered as a decomposition of an image on the base of functions bounded or localized in space; and on the other, the wavelet transforms are related to subband filtering and resampling.

The Intel IPP for image processing contains one-level discrete wavelet decomposition and reconstruction functions. It also provides the necessary interface for initialization and deallocation of the transform context structure. [Table 13-1](#) lists all functions of this group:

**Table 13-1 Image Wavelet Transform Functions**

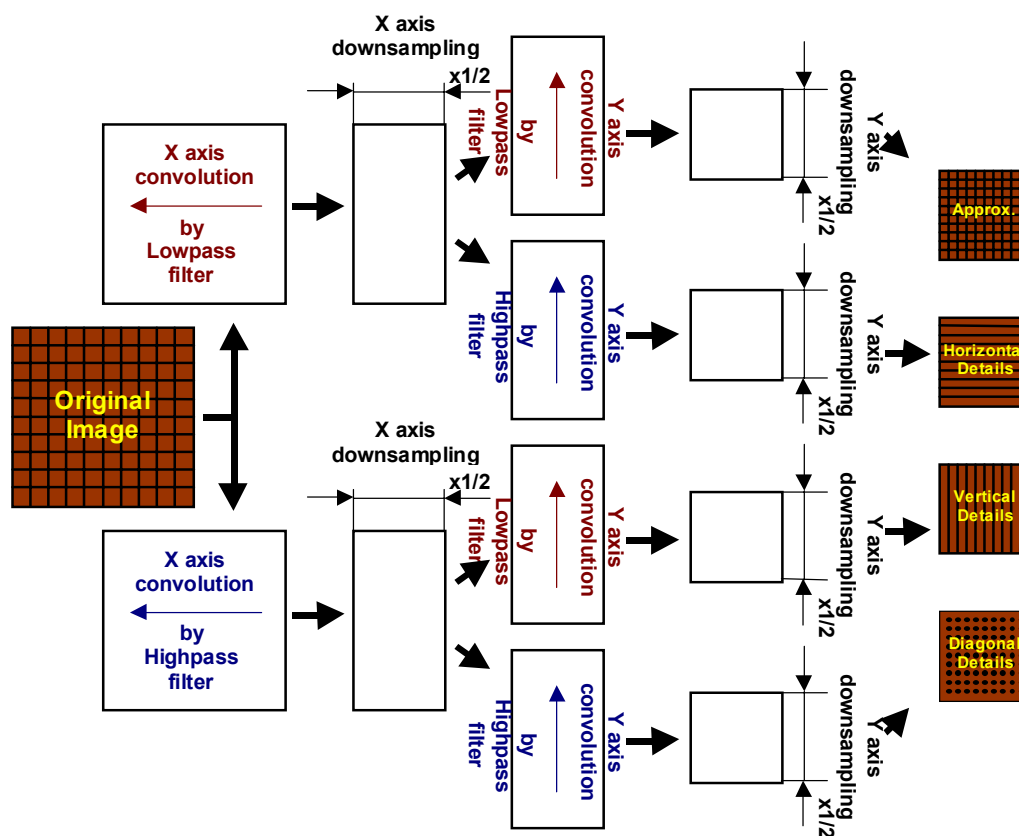
Function Base Name	Operation
<a href="#">WTFwdInitAlloc</a>	Allocates memory and initializes the forward wavelet transform context structure
<a href="#">WTFwdFree</a>	Deallocates memory allocated for a forward wavelet transform context structure
<a href="#">WTFwdGetBufSize</a>	Determines the size of external work buffer for a forward wavelet transform
<a href="#">WTFwd</a>	Performs one-level wavelet decomposition of an image
<a href="#">WTInvInitAlloc</a>	Allocates memory and initializes the inverse wavelet transform context structure
<a href="#">WTInvFree</a>	Deallocates memory allocated for an inverse wavelet transform context structure
<a href="#">WTInvGetBufSize</a>	Determines the size of external work buffer for an inverse wavelet transform
<a href="#">WTInv</a>	Performs one-level wavelet reconstruction of an image

You can set the wavelet transform type by specifying the appropriate filter taps in the initialization function.

Note that Intel IPP supports only one-dimensional finite impulse response filters for separable convolution.

The Intel IPP wavelet decomposition and reconstruction functions use fast polyphase algorithm, which is equivalent to traditional application of separable convolution and dyadic resampling in different order. The figure below shows the equivalent algorithm of wavelet-based image decomposition:

**Figure 13-1**    **Equivalent Scheme of Wavelet Decomposition Algorithm**



Decomposition operation applied to a source image produces four output images of equal size: approximation image, horizontal detail image, vertical detail image, and diagonal detail image.

These decomposition components have the following meaning:

- The ‘approximation’ image is obtained by vertical and horizontal lowpass filtering.
- The ‘horizontal detail’ image is obtained by vertical highpass and horizontal lowpass filtering.
- The ‘vertical detail’ image is obtained by vertical lowpass and horizontal highpass filtering.
- The ‘diagonal detail’ image is obtained by vertical and horizontal highpass filtering.

The above image names are used in this manual for identification convenience only.

The wavelet-based image reconstruction can be represented by a sequence of separate convolution and dyadic upsampling.

The reconstruction function uses four input images that are the same as those resulting from the decomposition operation.

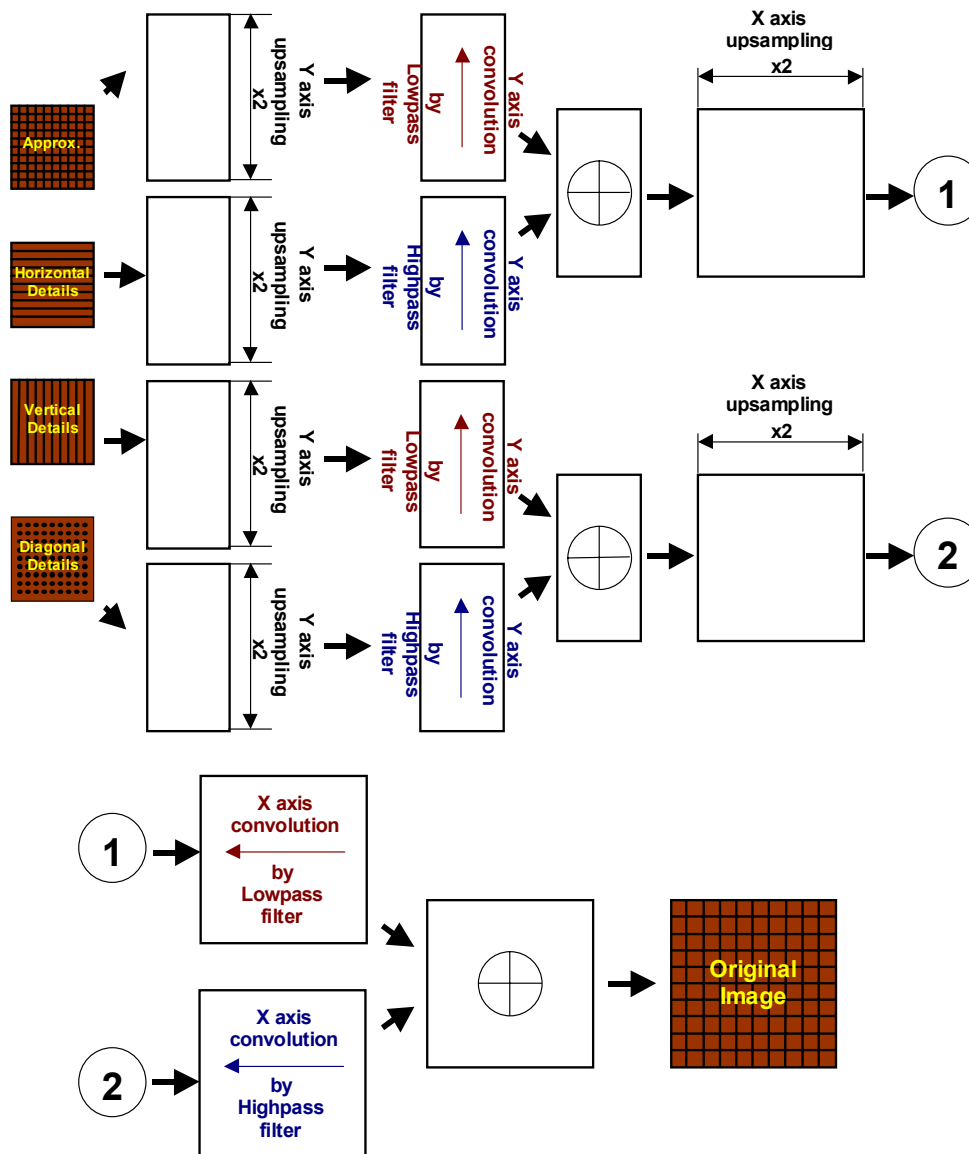
[Figure 13-2](#) shows the equivalent algorithm of wavelet reconstruction of an image.

Wavelet transform functions support regions of interest (ROI) in the images. However, these functions do not perform internally any border extensions of image ROI data.

It means that source images must already contain all border data that are necessary for convolution operations. See descriptions of [ippiWTFwd](#) and [ippiWTInv](#) functions for detailed information on how to calculate extended image border sizes.

The use of the Intel IPP for wavelet transform is demonstrated in the *Wavelet Transform Sample*. In addition to this sample many solutions and hints for use of Intel IPP in wavelet applications can be found in the *JPEG 2000 Encode-Decode Sample*. Both samples can be downloaded from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>.

Figure 13-2 Equivalent Scheme of Wavelet Reconstruction Algorithm



---

## WTFwdInitAlloc

*Allocates memory and initializes the forward wavelet transform context structure.*

---

### Syntax

```
IppStatus ippiWTFwdInitAlloc_32f_C1R (IppiWTFwdSpec_32f_C1R** pSpec,
    const Ipp32f* pTapsLow, int lenLow, int anchorLow,
    const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);

IppStatus ippiWTFwdInitAlloc_32f_C3R (IppiWTFwdSpec_32f_C3R** pSpec,
    const Ipp32f* pTapsLow, int lenLow, int anchorLow,
    const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

### Parameters

<i>pSpec</i>	Pointer to pointer to a new allocated and initialized forward DWT context structure.
<i>pTapsLow</i>	Pointer to lowpass filter taps.
<i>lenLow</i>	Length of the lowpass filter.
<i>anchorLow</i>	Anchor position of the lowpass filter.
<i>pTapsHigh</i>	Pointer to highpass filter taps.
<i>lenHigh</i>	Length of the highpass filter.
<i>anchorHigh</i>	Anchor position of the highpass filter.

### Description

The function `ippiWTFwdInitAlloc` is declared in the `ippi.h` file. This function allocates memory for the context structure *pSpec* of a one-level wavelet decomposition and initializes this structure.

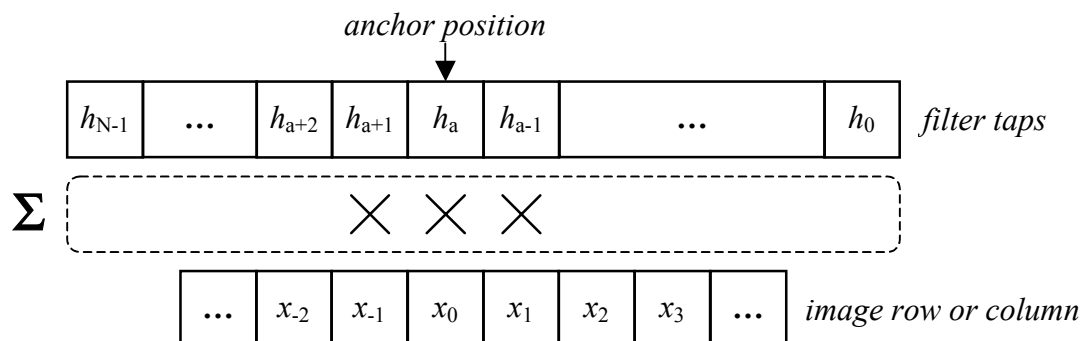
The forward wavelet transform context structure is assumed here to contain parameters of a wavelet filter bank used for image decomposition.

The filter bank consists of two analysis filters and includes the low pass decomposition filter (or coarse filter) and the high pass decomposition filter (or detail filter).

For the initialization to take place, your application should provide sets of coefficients *pTapsLow*, *pTapsHigh* and anchor positions *anchorLow*, *anchorHigh* for two analysis filters. The anchor value sets the initial leftmost filter position relative to image row or column as shown in the figure below:

**Figure 13-3 Anchor Value and Initial Filter Position for Wavelet Decomposition**

---



In this scheme, *a* stands for anchor value, *N* is filter length,  $x_0$  is the starting pixel of the processed row or column, and  $x_{-1}$ ,  $x_{-2}$ , ... are the additional border pixels that are needed for calculations. The anchor value and filter length completely determine right, left, top, and bottom border sizes for the source image used in decomposition. The corresponding C-language expressions to calculate border sizes are given in the description of [ippiWTFwd](#) function.

Once allocated and initialized context structure can be used in several threads of calculations as well as in a number of wavelet decomposition levels.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pTapsLow</i> , <i>pTapsHigh</i> , or <i>pSpec</i> pointer is NULL..
<code>ippStsSizeErr</code>	Indicates an error condition if filter length <i>lenLow</i> or <i>lenHigh</i> is less than 2.
<code>ippStsAnchorErr</code>	Indicates an error condition if anchor position <i>anchorLow</i> or <i>anchorHigh</i> is less than zero.
<code>ippStsMemAllocErr</code>	Indicates an error condition in case of memory allocation failure for the context structure.

---

## WTFwdFree

*Deallocates memory allocated for a forward wavelet transform context structure.*

---

### Syntax

```
IppStatus ippiWTFwdFree_32f_C1R (IppiWTFwdSpec_32f_C1R* pSpec);  
IppStatus ippiWTFwdFree_32f_C3R (IppiWTFwdSpec_32f_C3R* pSpec);
```

### Parameters

*pSpec*                      Pointer to an allocated and initialized forward DWT context structure.

### Description

The function `ippiWTFwdFree` is declared in the `ippi.h` file. This function deallocates memory that was previously allocated for a forward wavelet transform context structure *pSpec*. You should call `ippiWTFwdFree` to free each context structure that was allocated and initialized by [`ippiWTFwdInitAlloc`](#) function.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSpec</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid context structure is passed.

---

## WTFwdGetBufSize

*Determines the size of external work buffer for a forward wavelet transform.*

---

### Syntax

```
IppStatus ippiWTFwdGetBufSize_C1R(const IppiWTFwdSpec_32f_C1R* pSpec,  
int* pSize);
```



```
IppStatus ippiWTFwdGetBufSize_C3R(const IppiWTFwdSpec_32f_C3R* pSpec,
    int* pSize);
```

### Parameters

<i>pSpec</i>	Pointer to an allocated and initialized forward DWT context structure.
<i>pSize</i>	Pointer to a variable that will receive the size of work buffer required for forward wavelet transform.

### Description

The function `ippiWTFwdGetBufSize` is declared in the `ippi.h` file. This function computes the size in bytes of the work buffer required for the forward wavelet transform function [ippiWTFwd](#) to operate.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid context structure is passed.

---

## WTFwd

*Performs one-level wavelet decomposition of an image.*

---

### Syntax

```
IppStatus ippiWTFwd_32f_C1R (const Ipp32f* pSrc, int srcStep, Ipp32f*
    pApproxDst, int approxStep, Ipp32f* pDetailXDst, int detailXStep,
    Ipp32f* pDetailYDst, int detailYStep, Ipp32f* pDetailXYDst, int
    detailXYStep, IppiSize dstRoiSize, const IppiWTFwdSpec_32f_C1R*
    pSpec, Ipp8u* pBuffer);
```

```
IppStatus ippiWTFwd_32f_C3R (const Ipp32f* pSrc, int srcStep, Ipp32f*
    pApproxDst, int approxStep, Ipp32f* pDetailXDst, int detailXStep,
    Ipp32f* pDetailYDst, int detailYStep, Ipp32f* pDetailXYDst, int
    detailXYStep, IppiSize dstRoiSize, const IppiWTFwdSpec_32f_C3R*
    pSpec, Ipp8u* pBuffer);
```

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image buffer.
<i>pApproxDst</i>	Pointer to ROI of the destination approximation image.
<i>approxStep</i>	Distance in bytes between starts of consecutive lines in the approximation image buffer.
<i>pDetailXDst</i>	Pointer to ROI of the destination horizontal detail image.
<i>detailXStep</i>	Distance in bytes between starts of consecutive lines in the horizontal detail image buffer.
<i>pDetailYDst</i>	Pointer to ROI of the destination vertical detail image.
<i>detailYStep</i>	Distance in bytes between starts of consecutive lines in the vertical detail image buffer.
<i>pDetailXYDst</i>	Pointer to ROI of the destination diagonal detail image.
<i>detailXYStep</i>	Distance in bytes between starts of consecutive lines in the diagonal detail image buffer.
<i>dstRoiSize</i>	Size of the ROI in pixels for all destination images.
<i>pSpec</i>	Pointer to the allocated and initialized forward DWT context structure.
<i>pBuffer</i>	Pointer to the allocated buffer for intermediate operations

## Description

The function `ippiWTFwd` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function performs one-level wavelet decomposition of a source image pointed to by *pSrc* argument into four destination subimages pointed to by *pApproxDst*, *pDetailXDst*, *pDetailYDst*, and *pDetailXYDst*. See [Figure 13-1](#) for the equivalent algorithm of `ippiWTFwd` function operation.

Wavelet parameters are contained in the forward transform context structure pointed to by *pSpec*, which must be allocated and initialized by [ippiWTFwdInitAlloc](#) function before calling *ippiWTFwd*.

The decomposition function needs a buffer for temporary calculations, which is pointed to by *pBuffer* argument. You can call *ippiWTFwdGetBufSize* function to determine the required buffer size. Note that the buffer size does not depend upon the size of processed images, so the same buffer can be used for decomposing images of different size. However, it is not recommended to use the same buffer in different threads in a multithreaded mode. For better performance, use an aligned memory location for this buffer, which can be allocated by means of [ippsMalloc](#) function.

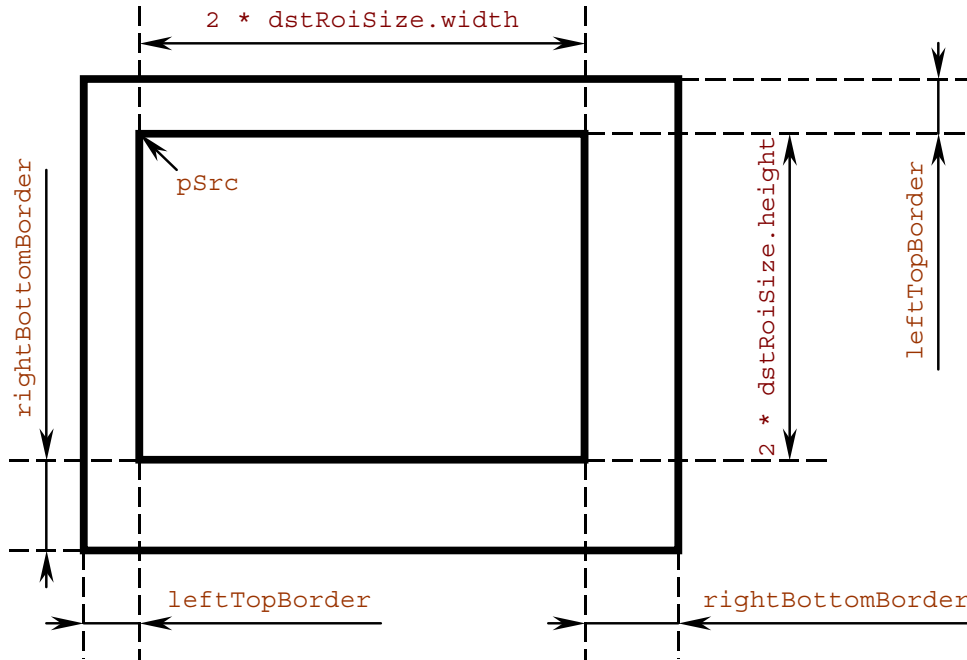
Note that *pSrc* argument points to memory location of the source image rectangular ROI of size *srcWidth* by *srcHeight* which is uniquely determined by the size of destination ROI as:

```
srcWidth  = 2 * dstRoiSize.width
srcHeight = 2 * dstRoiSize.height
```

Thus, source image ROI does not include border pixels necessary to compute some destination pixels. It means that prior to using *ippiWTFwd* function your application program have to apply some border extension model (symmetrical, wraparound or another) to the source image ROI through filling of neighboring memory locations.

As a result, the size of memory block allocated for the source image should be extended to accommodate for added border pixels outside ROI formal boundaries.

[Figure 13-4](#) schematically shows the source image ROI and extended image area.

**Figure 13-4 Extended Source Image for Wavelet Decomposition**

Use the following C-language expressions to calculate extended image border sizes:

```
int leftBorderLow  = lenLow  - 1 - anchorLow;
int leftBorderHigh = lenHigh - 1 - anchorHigh;
int rightBorderLow = lenLow  - 2 - leftBorderLow;
int rightBorderHigh = lenHigh - 2 - leftBorderHigh;
int leftTopBorder  = IPP_MAX(leftBorderLow, leftBorderHigh);
int rightBottomBorder = IPP_MAX(rightBorderLow, rightBorderHigh);
```

See the description of [ippiWTFwdInitAlloc](#) function for the explanation of parameters used.

Note that the left and top borders have equal size. The same holds for the right and bottom borders which have equal size too.

The size of the source image area extended by border pixels can be defined as

```
srcWidthWithBorders = srcWidth + leftTopBorder + rightBottomBorder;  
srcHeightWithBorders = srcHeight + leftTopBorder + rightBottomBorder;
```

All of the four destination images have equal size specified by *dstRoiSize* argument.

Conversely, if you need to perform a wavelet reconstruction of the full size source image from the component images obtained by decomposition, you should use extended component images for the reconstruction pass. See the discussion of [ippiWTInv](#) function for details.

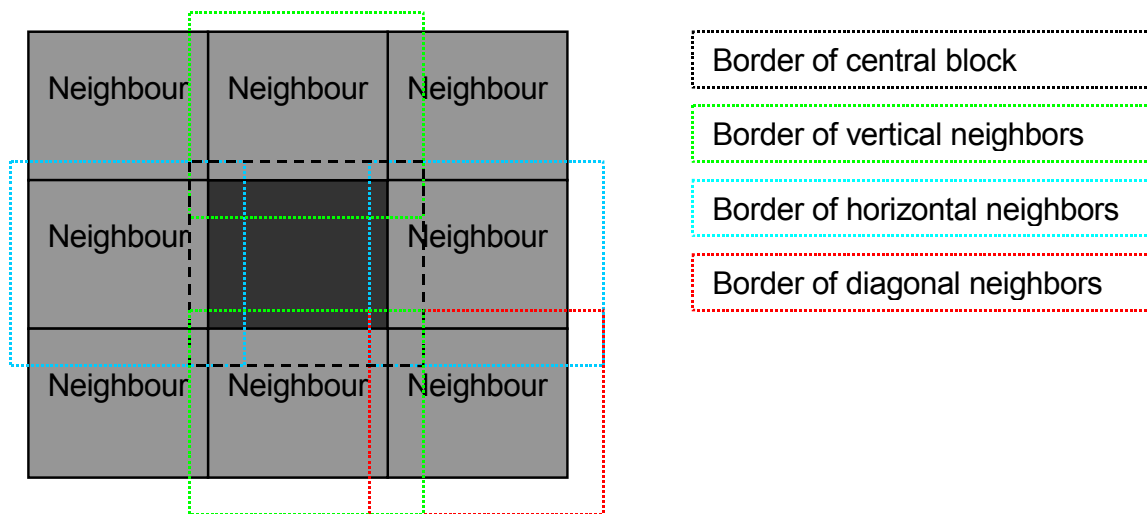
The ROI concept used in wavelet transform functions can be applied to processing large images by blocks or ‘tiles’. To accomplish this, you should subdivide the source image into overlapping blocks in the following way:

- Main part (ROI) of each block is adjacent to neighboring blocks and has no intersection with neighbor’s ROIs;
- Extended borders of each block overlap with ROIs of neighboring blocks.

This subdivision scheme is illustrated in the figure below.

**Figure 13-5 Image Division into Blocks with Overlapping Borders**

---



## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if step through any buffer is less than or equal to zero.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid context structure is passed.

---

## WTInvInitAlloc

*Allocates memory and initializes the inverse wavelet transform context structure.*

---

### Syntax

```
IppStatus ippWTInvInitAlloc_32f_C1R (IppiWTInvSpec_32f_C1R** pSpec,
    const Ipp32f* pTapsLow, int lenLow, int anchorLow,
    const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);

IppStatus ippWTInvInitAlloc_32f_C3R (IppiWTInvSpec_32f_C3R** pSpec,
    const Ipp32f* pTapsLow, int lenLow, int anchorLow,
    const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

### Arguments:

<code>pSpec</code>	Pointer to pointer to a new allocated and initialized inverse DWT context structure.
<code>pTapsLow</code>	Pointer to lowpass filter taps.
<code>lenLow</code>	Length of the lowpass filter.
<code>anchorLow</code>	Anchor position of the lowpass filter.
<code>pTapsHigh</code>	Pointer to highpass filter taps.
<code>lenHigh</code>	Length of the highpass filter.

*anchorHigh*

Anchor position of the highpass filter.

### Description

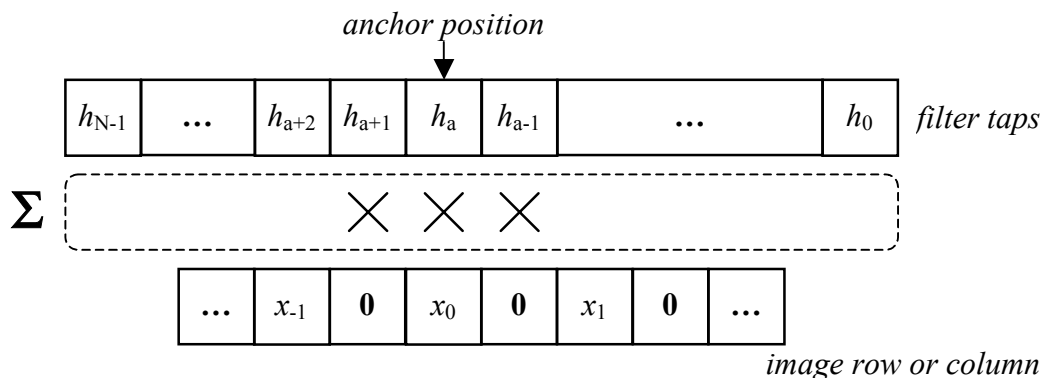
The function `ippiWTInvInitAlloc` is declared in the `ippi.h` file. This function allocates memory for the context structure *pSpec* of a one-level wavelet reconstruction and initializes this structure.

The inverse wavelet transform context structure is assumed here to contain parameters of a wavelet filter bank used for image reconstruction. The filter bank consists of two synthesis filters and includes the low pass reconstruction filter (or coarse filter) and the high pass reconstruction filter (or detail filter).

For the initialization to take place, your application should provide sets of coefficients *pTapsLow*, *pTapsHigh* and anchor positions *anchorLow*, *anchorHigh* for two synthesis filters. The anchor value sets the initial leftmost filter position relative to image row or column as shown in the figure below:

**Figure 13-6 Anchor Value and Initial Filter Position for Wavelet Reconstruction**

---



Here *a* stands for anchor value, *N* is filter length,  $x_0$  is the starting pixel of the processed row or column, and  $x_{-1}, x_{-2}, \dots$  are the additional border pixels that are needed for calculations. As seen from this figure, anchor position is specified relative to upsampled source data. The anchor value and filter length completely determine right, left, top, and bottom border sizes for source images used in reconstruction. The corresponding C-language expressions to calculate border sizes are given in the description of [ippiWTInv](#) function.

Once allocated and initialized context structure can be used in several threads of calculations as well as in a number of wavelet reconstruction levels.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pTapsLow</i> , <i>pTapsHigh</i> , or <i>pSpec</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if filter length <i>lenLow</i> or <i>lenHigh</i> is less than 2.
<code>ippStsAnchorErr</code>	Indicates an error condition if anchor position <i>anchorLow</i> or <i>anchorHigh</i> is less than zero.
<code>ippStsMemAllocErr</code>	Indicates an error condition if memory allocation fails.

---

## WTInvFree

*Deallocates memory allocated for an inverse wavelet transform context structure.*

---

### Syntax

```
IppStatus ippWTInvFree_32f_C1R (IppiWTInvSpec_32f_C1R* pSpec);
IppStatus ippWTInvFree_32f_C3R (IppiWTInvSpec_32f_C3R* pSpec);
```

### Parameters

*pSpec*                      Pointer to an allocated and initialized inverse DWT context structure.

### Description

The function `ippWTInvFree` is declared in the `ippi.h` file. This function deallocates memory that was previously allocated for an inverse wavelet transform context structure *pSpec*. You should call `ippWTInvFree` to free each context structure that was allocated and initialized by [`ippWTInvInitAlloc`](#) function.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--



<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSpec</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid context structure is passed.

---

## WTInvGetBufSize

*Determines the size of external work buffer for an inverse wavelet transform.*

---

### Syntax

```
IppStatus ippWTInvGetBufSize_C1R( const IppiWTInvSpec_32f_C1R* pSpec,
    int* pSize );
IppStatus ippWTInvGetBufSize_C3R( const IppiWTInvSpec_32f_C3R* pSpec,
    int* pSize );
```

### Parameters

<i>pSpec</i>	Pointer to an allocated and initialized inverse DWT context structure.
<i>pSize</i>	Pointer to a variable that will receive the size of work buffer for inverse wavelet transform.

### Description

The function `ippWTInvGetBufSize` is declared in the `ippi.h` file. This function computes the size in bytes of the work buffer that is required for the inverse wavelet transform function [ippWTInv](#) to operate.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid context structure is passed.

## WTInv

*Performs one-level wavelet reconstruction of an image.*

### Syntax

```
IppStatus ippiWTInv_32f_C1R(const Ipp32f* pApproxSrc, int approxStep, const
    Ipp32f* pDetailXSrc, int detailXStep, const Ipp32f* pDetailYSrc, int
    detailYStep, const Ipp32f* pDetailXYSrc, int detailXYStep, IppiSize
    srcRoiSize, Ipp32f* pDst, int dstStep, const IppiWTInvSpec_32f_C1R* pSpec,
    Ipp8u* pBuffer);
```

```
IppStatus ippiWTInv_32f_C3R(const Ipp32f* pApproxSrc, int approxStep, const
    Ipp32f* pDetailXSrc, int detailXStep, const Ipp32f* pDetailYSrc, int
    detailYStep, const Ipp32f* pDetailXYSrc, int detailXYStep, IppiSize
    srcRoiSize, Ipp32f* pDst, int dstStep, const IppiWTInvSpec_32f_C3R* pSpec,
    Ipp8u* pBuffer);
```

### Parameters

<i>pApproxSrc</i>	Pointer to ROI of the source approximation image.
<i>approxStep</i>	Distance in bytes between starts of consecutive lines in the approximation image buffer.
<i>pDetailXSrc</i>	Pointer to ROI of the source horizontal detail image.
<i>detailXStep</i>	Distance in bytes between starts of consecutive lines in the horizontal detail image buffer.
<i>pDetailYSrc</i>	Pointer to ROI of the source vertical detail image.
<i>detailYStep</i>	Distance in bytes between starts of consecutive lines in the vertical detail image buffer.
<i>pDetailXYSrc</i>	Pointer to ROI of the source diagonal detail image.
<i>detailXYStep</i>	Distance in bytes between starts of consecutive lines in the diagonal detail image buffer.
<i>srcRoiSize</i>	Size of ROI in pixels for all source images.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image buffer.

<i>pSpec</i>	Pointer to the allocated and initialized inverse DWT context structure.
<i>pBuffer</i>	Pointer to the allocated buffer for intermediate operations.

## Description

The function `ippiWTInv` is declared in the `ippi.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function performs wavelet reconstruction of the output image *pDst* from the four component images.

See [Figure 13-1](#) for the equivalent algorithm of `ippiWTInv` function operation. Wavelet parameters are contained in inverse transform context structure pointed to by *pSpec*, which must be allocated and initialized by [ippiWTInvInitAlloc](#) function prior to calling `ippiWTInv`. The reconstruction function needs a buffer for temporary calculations, which is pointed to by *pBuffer* argument. You can call `ippiWTInvGetBufSize` function to determine the required buffer size. Note that the buffer size does not depend upon the size of processed images, so the same buffer can be used for reconstructing images of different size. However, it is not recommended to use the same buffer in different threads in a multithreaded mode. For better performance, use an aligned memory location for this buffer, which can be allocated by means of [ippsMalloc](#) function.

The pointer arguments *pApproxSrc*, *pDetailXSrc*, *pDetailYSrc*, and *pDetailXYSrc* point to ROIs of source images excluding borders. All source ROIs have the same size *srcRoiSize*, while the destination image size is uniquely determined from the following relations:

```
dstWidth  = 2 * srcRoiSize.width;
dstHeight = 2 * srcRoiSize.height;
```

As source ROIs do not include border pixels necessary to compute some destination pixels, your application program have to apply a border extension model (symmetrical, wraparound or another) to ROIs of all source images by means of filling the neighboring memory locations.

Note that border sizes may be different for different source images. Use the following C-language expressions to calculate extended image border sizes:

```
int leftBorderLow   = (lenLow  - 1 - anchorLow) / 2;
int leftBorderHigh  = (lenHigh - 1 - anchorHigh) / 2;

int rightBorderLow  = (anchorLow + 1) / 2;
int rightBorderHigh = (anchorHigh + 1) / 2;

int apprLeftBorder  = leftBorderLow;
int apprRightBorder = rightBorderLow;
int apprTopBorder   = leftBorderLow;
int apprBottomBorder = rightBorderLow;
```

```

int detxLeftBorder    = leftBorderLow;
int detxRightBorder   = rightBorderLow;
int detxTopBorder     = leftBorderHigh;
int detxBottomBorder  = rightBorderHigh;

int detyLeftBorder    = leftBorderHigh;
int detyRightBorder   = rightBorderHigh;
int detyTopBorder     = leftBorderLow;
int detyBottomBorder  = rightBorderLow;

int detxyLeftBorder   = leftBorderHigh;
int detxyRightBorder  = rightBorderHigh;
int detxyTopBorder    = leftBorderHigh;
int detxyBottomBorder = rightBorderHigh;

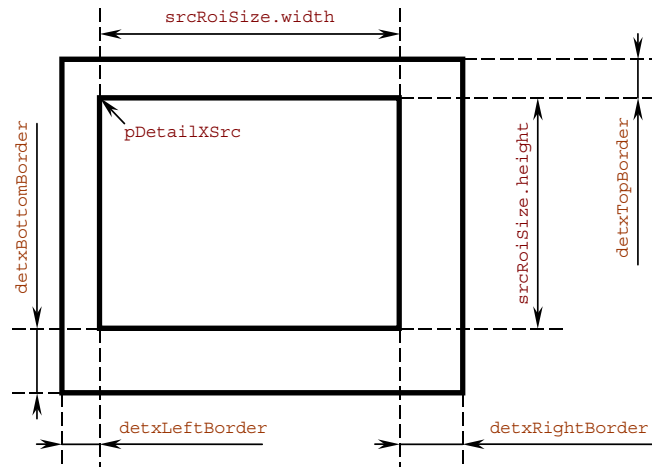
```

See the description of [ippiWTInvInitAlloc](#) function for the explanation of parameters used.

As seen from the above relations, left and top borders always have equal size only for approximation and diagonal detail images. Right and bottom borders also have equal size only for approximation and diagonal detail images. Thus, the size of memory block allocated for each source image should be extended to accommodate for added border pixels outside ROI formal boundaries.

As an illustration, ROI and extended image area for the horizontal detail source image are shown in the figure below:

**Figure 13-7 Extended Horizontal Detail Source Image for Wavelet Reconstruction**



Sizes of source images extended by border pixels can be calculated as follows:

```
apprWidthWithBorders  = srcWidth  + apprLeftBorder + apprRightBorder;  
apprHeightWithBorders = srcHeight + apprTopBorder  + apprBottomBorder;  
  
detxWidthWithBorders  = srcWidth  + detxLeftBorder + detxRightBorder;  
detxHeightWithBorders = srcHeight + detxTopBorder  + detxBottomBorder;  
  
detyWidthWithBorders  = srcWidth  + detyLeftBorder + detyRightBorder;  
detyHeightWithBorders = srcHeight + detyTopBorder  + detyBottomBorder;  
  
detxyWidthWithBorders = srcWidth  + detxyLeftBorder + detxyRightBorder;  
detxyHeightWithBorders = srcHeight + detxyTopBorder  + detxyBottomBorder;
```

The ROI concept can be used to reconstruct large images by blocks or ‘tiles’.

To accomplish this, you should subdivide each of the source images into blocks with overlapping borders, similar to what is considered in the [ippiWTFwd](#) function description. Each of the four components must be subdivided into the same pattern of rectangular blocks.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if step through any buffer is less than or equal to zero.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid context structure is passed.

This chapter provides a background for the computer vision concepts used in the Intel IPP software as well as detailed descriptions of the Intel IPP functions for computer vision. The Intel IPP functions are combined in groups by their functionality. Each group of functions is described in a separate section.

**Table 14-1 Intel IPP Functions for Computer Vision Applications**

Function Name	Description
<b>Feature Detection Functions</b>	
<a href="#">CannyGetSize</a>	Pre-calculates the size of a temporary buffer for the function <code>ippiCanny</code> .
<a href="#">Canny</a>	Finds edges in the image.
<a href="#">EigenValsVecsGetBufferSize</a>	Pre-calculates the size of a temporary buffer for the function <code>ippiEigenValsVecs</code> .
<a href="#">EigenValsVecs</a>	Calculates eigen values and eigen vectors of image blocks for corner detection.
<a href="#">MinEigenValGetBufferSize</a>	Pre-calculates the size of a temporary buffer for the function <code>ippiCornerMinEigenVal</code> .
<a href="#">MinEigenVal</a>	Calculates the minimal eigen value of image blocks for corner detection.
<b>Distance Transform Functions</b>	
<a href="#">DistanceTransform</a>	Performs the distance transform operation.
<a href="#">GetDistanceTransformMask</a>	Calculates metrics for distance transform.
<b>Image Gradients</b>	
<a href="#">GradientColorToGray</a>	Converts a color gradient image to grayscale.
<b>Flood Fill Functions</b>	
<a href="#">FloodFillGetSize</a>	Calculates size of temporary buffer for flood filling operation.
<a href="#">FloodFillGetSize_Grad</a>	Calculates size of temporary buffer for the gradient flood filling.

**Table 14-1 Intel IPP Functions for Computer Vision Applications (continued)**

Function Name	Description
<a href="#"><u>FloodFill</u></a>	Fills the connected area of the same pixel values on the image with a new value, considering 4 or 8 neighbor values of the pixel.
<a href="#"><u>FloodFill_Grad</u></a>	Fills the connected area of the pixel values within a small threshold on the image with a new value, considering 4 or 8 neighbor values of the pixel.
<a href="#"><u>FloodFill_Range</u></a>	Fills the connected region of the pixel values within a specified range on the image with a new value, considering 4 neighbor values of the pixel
<b>Motion Templates Functions</b>	
<a href="#"><u>UpdateMotionHistory</u></a>	Updates the motion history image using the motion silhouette at a given timestamp.
<a href="#"><u>OpticalFlowPyrLKInitAlloc</u></a>	Allocates memory and initializes a structure for optical flow calculation.
<a href="#"><u>OpticalFlowPyrLKFree</u></a>	Frees memory allocated for the optical flow structure.
<a href="#"><u>OpticalFlowPyrLK</u></a>	Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.
<b>Pyramids Functions</b>	
<a href="#"><u>PyrDownGetBufSize</u></a>	Calculates the size of a temporary buffer used by the function PyrDown.
<a href="#"><u>PyrUpGetBufSize</u></a>	Calculates the size of a temporary buffer used by the function PyrUp.
<a href="#"><u>PyrDown</u></a>	Applies the Gaussian to an image, then performs down-sampling.
<a href="#"><u>PyrUp</u></a>	Performs up-sampling of an image, then applies the Gaussian multiplied by 4.
<b>General Pyramids Functions</b>	
<a href="#"><u>PyramidInitAlloc</u></a>	Allocates memory and initializes a pyramid structure.
<a href="#"><u>PyramidFree</u></a>	Frees memory allocated for the pyramid structure.
<a href="#"><u>PyramidLayerDownInitAlloc</u></a>	Allocates memory and initializes a structure for creating a lower pyramid layer.
<a href="#"><u>PyramidLayerUpInitAlloc</u></a>	Allocates memory and initializes a structure for creating an upper pyramid layer.
<a href="#"><u>PyramidLayerDownFree</u></a>	Frees memory allocated for the lower pyramid layer structure.
<a href="#"><u>PyramidLayerUpFree</u></a>	Frees memory allocated for the upper pyramid layer structure.
<a href="#"><u>GetPyramidDownROI</u></a>	Computes the size of the lower pyramid layer.
<a href="#"><u>GetPyramidUpROI</u></a>	Computes the size of the upper pyramid layer.

**Table 14-1 Intel IPP Functions for Computer Vision Applications (continued)**

Function Name	Description
<a href="#">PyramidLayerDown</a>	Creates a lower pyramid layer.
<a href="#">PyramidLayerUp</a>	Creates an upper pyramid layer.
<b>Pattern Recognition Functions</b>	
<a href="#">HaarClassifierInitAlloc</a>	Allocates memory and initializes the structure for standard Haar classifiers.
<a href="#">TiltedHaarClassifierInitAlloc</a>	Allocates memory and initializes the structure for tilted Haar classifiers.
<a href="#">HaarClassifierFree</a>	Frees memory allocated for the Haar classifier structure.
<a href="#">GetHaarClassifierSize</a>	Returns the size of the Haar classifier.
<a href="#">TiltHaarFeatures</a>	Modifies a Haar classifier by tilting specified features.
<a href="#">ApplyHaarClassifier</a>	Applies a Haar classifier to an image.
<a href="#">ApplyMixedHaarClassifier</a>	Applies a mixed Haar classifier to an image.
<b>Camera Calibration and 3D Reconstruction</b>	
<a href="#">UndistortGetSize</a>	Computes the size of the external buffer.
<a href="#">UndistortRadial</a>	Corrects radial distortions of the single image.
<a href="#">CreateMapCameraUndistort</a>	Creates look-up tables of coordinates of corrected image

## Using `ippiAdd` for Background Differencing

This section describes functions that help build a statistical model of a background. This model can be used to subtract the background from an image.

Here, the term “background” stands for a set of motionless image pixels – that is, pixels that do not belong to any object, moving in front of the camera. The definition of background can vary if considered in other techniques of object extraction. For example, if the depth map of the scene can be obtained, e.g., with the help of stereo, background can be determined as static parts of the scene that are located far enough from the camera.

The simplest background model assumes that every background pixel brightness varies independently, according to normal distribution. To calculate the characteristics of the background, several dozens of frames, as well as their squares, can be accumulated. That is, for every pixel location we find the sum of pixel values in this location  $S(x, y)$ , using the function [ippiAdd](#), and the sum of squares of the values  $Sq(x, y)$ , using the function [ippiAddSquare](#). Then mean is calculated as

$$m(x, y) = \frac{S(x, y)}{N} \quad , \text{ where } N \text{ is the number of collected frames, and standard deviation as}$$



$$stddev(x, y) = \sqrt{\frac{Sq(x, y)}{N} - \left(\frac{S(x, y)}{N}\right)^2}$$

After that, the pixel in a certain pixel location within a certain frame is considered as belonging to a moving object, if the condition  $abs(p(x, y) - m(x, y)) < C * stddev(x, y)$ , where  $C$  is a constant, is met. If  $C$  is equal to 3, it satisfies the “three sigmas” rule. To obtain such background model, objects should be put away from the camera for a few seconds, so that the whole image from the camera represents the subsequent background observation.

Adapting the background differencing model to changes in lighting conditions and background scenes, for example, when the camera moves or an object passes behind the front object, can improve the described technique.

The mean brightness can be calculated through replacing the simple average with the running average found by using the function [ippiAddWeighted](#). Also, several techniques can be used to identify moving scene parts and exclude them while accumulating background information. These techniques include change detection (see the functions [ippiAbsDiff](#) in chapter 5 and [ippiThreshold](#) in chapter 7), optical flow, and some other operations.

Relevant addition functions used for background differencing include:

Add\_8u32f\_C1IR, Add\_8s32f\_C1IR, Add\_32f\_C1IR,  
Add\_8u32f\_C1IMR, Add\_8s32f\_C1IMR, Add\_32f\_C1IMR (see [Add](#)),

and also all flavors of [AddSquare](#), [AddProduct](#), and [AddWeighted](#).

## Feature Detection Functions

This section describes feature detection functions. The set of Sobel derivative filters is generally used to find edges, ridges, and blobs, especially in case of scale-space images, for example, pyramids.

The following naming conventions are used in the equations described below:

- $D_x$  and  $D_y$  are the first  $x$  and  $y$  derivatives, respectively.
- $D_{xx}$  and  $D_{yy}$  are the second  $x$  and  $y$  derivatives, respectively.
- $D_{xy}$  is the partial  $x$  and  $y$  derivative.
- $D_{xxx}$  and  $D_{yyy}$  are the third  $x$  and  $y$  derivatives, respectively.
- $D_{xxy}$  and  $D_{xyy}$  are the third partial  $x$  and  $y$  derivatives.

## Corner Detection

The Sobel and Sharr first derivative operators are to be used to take the  $x$  and  $y$  derivatives of an image. Then a small region of interest (ROI) is to be defined to detect corners in.

A 2x2 matrix of sums of the  $x$  and  $y$  derivatives is created as follows:

$$C = \begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix} .$$

Solving  $\det(C - \lambda I) = 0$ , where  $\lambda$  is a column vector of the eigen values and  $I$  is the identity matrix, gives the eigen values. For the 2x2 matrix of the equation above, the solutions may be written in closed form:

$$\lambda = \frac{\Sigma D_x^2 + \Sigma D_y^2 \pm \sqrt{(\Sigma D_x^2 + \Sigma D_y^2)^2 - 4 \cdot (\Sigma D_x^2 \Sigma D_y^2 - (\Sigma D_x D_y)^2)}}{2} .$$

If  $\lambda_1, \lambda_2 > t$ , where  $t$  is some threshold, then a corner is considered to be found at that location. This can be very useful for object or shape recognition.

## Canny Edge Detector

This subsection describes a classic edge detector proposed by J.Canny, see [[Canny86](#)]. The detector uses a grayscale image as an input and outputs a black-and-white image, where non-zero pixels mark detected edges. The algorithm consists of three stages described below.

### Stage 1: Differentiation

Assuming two-dimensional convolution, the image data are differentiated with respect to the directions  $x$  and  $y$ . The gradient of the surface of the convoluted image function in any direction is possible to compute from the known gradient in any two directions.

From the computed  $x$  and  $y$  gradient values, the magnitude and angle of the slope can be calculated from the hypotenuse and arctangent.



**NOTE.** The `ippiSobel` functions perform the first stage and Canny edge detector functions use their output.

### Stage 2: Non-Maximum Suppression

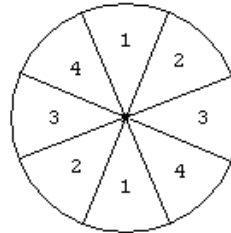
With the rate of intensity change found at each point in the image, edges must now be placed at the points of maximum, or rather non-maximum must be suppressed. A local maximum occurs at a peak in the gradient function, or alternatively where the derivative of the gradient function is set to zero. However, in this case it is preferable to suppress non-maximum perpendicular to the edge direction, rather than parallel to the edge direction, since the edge strength is expected to continue along an extended contour.

The algorithm starts off by reducing the angle of gradient to one of 4 sectors shown in [Figure 14-1](#). The algorithm passes 3×3 neighborhood across the magnitude array. At each point, the center element of neighborhood is compared with its two neighbors along line of the gradient given by the sector value.

If the central value is non-maximum, i.e., not greater than the neighbors, it is suppressed.

**Figure 14-1 Gradient Sectors**

---



---

### Stage 3: Edge Thresholding

The Canny operator uses the so-called “hysteresis” thresholding. Most thresholders use a single threshold limit, which means that if the edge values fluctuate above and below this value, the line appears broken. This phenomenon is commonly referred to as “streaking”. Hysteresis counters streaking by setting an upper and lower edge value limit. Considering a line segment, if a value lies above the upper threshold limit it is immediately accepted. If the value lies below the low threshold it is immediately rejected. Points which lie between the two limits are accepted if they

are connected to pixels which exhibit strong response. The likelihood of streaking is reduced drastically since the line segment points must fluctuate above the upper limit and below the lower limit for streaking to occur.

J.Canny recommends the ratio of high to low limit be in the range two or three to one, based on predicted signal-to-noise ratios.

---

## CannyGetSize

*Calculates size of temporary buffer for function `ippiCanny`.*

---

### Syntax

```
IppStatus ippiCannyGetSize(IppiSize roiSize, int* pBufSize);
```

### Parameters

<code>roiSize</code>	Size of the image ROI in pixels.
<code>pBufSize</code>	Pointer to the computed size of the temporary buffer.

### Description

The function `ippiCannyGetSize` is declared in the `ippcv.h` file. This function calculates the size of a temporary buffer for the function [ippiCanny](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pBufSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>pRoiSize</code> has a field with zero or negative value.

---

## Canny

*Implements Canny algorithm for edge detection.*

---

### Syntax

```
IppStatus ippiCanny_16s8u_C1R(Ipp16s* pSrcDx, int srcDxStep, Ipp16s* pSrcDy,  
    int srcDyStep, Ipp8u* pDstEdges, int dstEdgeStep, IppiSize roiSize, Ipp32f  
    lowThresh, Ipp32f highThresh, Ipp8u* pBuffer);  
  
IppStatus ippiCanny_32f8u_C1R(Ipp32f* pSrcDx, int srcDxStep, Ipp32f* pSrcDy,  
    int srcDyStep, Ipp8u* pDstEdges, int dstEdgeStep, IppiSize roiSize, Ipp32f  
    lowThresh, Ipp32f highThresh, Ipp8u* pBuffer);
```

### Parameters

<i>pSrcDx</i>	Pointer to the source image ROI <i>x</i> -derivative.
<i>srcDxStep</i>	Distance in bytes between starts of consecutive lines in the source image <i>pSrcDx</i> .
<i>pSrcDy</i>	Pointer to the source image ROI <i>y</i> -derivative.
<i>srcDyStep</i>	Distance in bytes between starts of consecutive lines in the source image <i>pSrcDy</i> .
<i>pDstEdges</i>	Pointer to the output array of the detected edges.
<i>dstEdgeStep</i>	Distance in bytes between starts of consecutive lines in the output image.
<i>roiSize</i>	Size of the source image ROI in pixels.
<i>lowThresh</i>	Lower threshold for edges detection.
<i>highThresh</i>	Upper threshold for edges detection.
<i>pBuffer</i>	Pointer to the pre-allocated temporary buffer.

### Description

The function `ippiCanny` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function finds edges in the source image ROI and stores them into the output image *pDstEdges* using the Canny algorithm. The function requires a temporary working buffer; its size should be computed previously by calling the function [ippiCannyGetSize](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>pRoiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcDxStep</code> , <code>srcDyStep</code> or <code>dstEdgeStep</code> is less than <code>roi.width * &lt;pixelSize&gt;</code> .
<code>ippStsBadArgErr</code>	Indicates an error when <code>lowThresh</code> is negative or <code>highThresh</code> is less than <code>lowThresh</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 2 for 16s images, and by 4 for 32f images.

## EigenValsVecsGetBufferSize

*Calculates size of temporary buffer for the function*  
`ippiEigenValsVecs`.

### Syntax

```
IppStatus ippiEigenValsVecsGetBufferSize_32f_C1R(IppiSize roiSize, int
    apertureSize, int avgWindow, int* pBufferSize);
IppStatus ippiEigenValsVecsGetBufferSize_8u32f_C1R(IppiSize roiSize, int
    apertureSize, int avgWindow, int* pBufferSize);
```

### Parameters

<code>roiSize</code>	Size of the source image ROI in pixels.
<code>apertureSize</code>	Size (pixels) of the derivative operator used by the function, possible values are 3 or 5.
<code>avgWindow</code>	Size of the blurring window in pixels, possible values are 3 or 5.
<code>pBufSize</code>	Pointer to the variable that returns the size of the temporary buffer.

## Description

The function `ippiEigenValsVecsGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function calculates the size of a temporary buffer to be used by the function [ippiEigenValsVecs](#).



---

**CAUTION.** The parameters `apertureSize` and `avgWindow` must be the same for both functions `ippiEigenValsVecsGetBufferSize` and `ippiEigenValsVecs`.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiWidth</code> has zero or negative value, or if <code>apertureSize</code> or <code>avgWindow</code> has an illegal value.

---

## EigenValsVecs

*Calculates eigen values and eigen vectors of image blocks for corner detection.*

---

## Syntax

```
ippiStatus ippiEigenValsVecs_8u32f_C1R(const Ipp8u* pSrc, int srcStep,
    Ipp32f* pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType
    kernType, int apertureSize, int avgWindow, Ipp8u* pBuffer);

ippiStatus ippiEigenValsVecs_32f_C1R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType
    kernType, int apertureSize, int avgWindow, Ipp8u* pBuffer);
```

## Parameters

`pSrc`                      Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.				
<i>pEigenVV</i>	Image to store the results.				
<i>eigStep</i>	Distance in bytes between starts of consecutive lines in the output image.				
<i>roiSize</i>	Size of the source image ROI in pixels.				
<i>kernType</i>	Specifies the type of kernel used to compute derivatives, possible values are: <table data-bbox="617 503 1185 589"> <tr> <td><code>ippKernelSobel</code></td><td>Sobel kernel 3x3 or 5x5;</td></tr> <tr> <td><code>ippKernelScharr</code></td><td>Scharr kernel 3x3.</td></tr> </table>	<code>ippKernelSobel</code>	Sobel kernel 3x3 or 5x5;	<code>ippKernelScharr</code>	Scharr kernel 3x3.
<code>ippKernelSobel</code>	Sobel kernel 3x3 or 5x5;				
<code>ippKernelScharr</code>	Scharr kernel 3x3.				
<i>apertureSize</i>	Size of the derivative operator in pixels, possible values are 3 or 5.				
<i>avgWindow</i>	Size of the blurring window in pixels, possible values are 3 or 5.				
<i>pBuffer</i>	Pointer to the temporary buffer.				

## Description

The function `ippiEigenValsVecs` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function takes a block around the pixel and computes the first derivatives  $D_x$  and  $D_y$ . This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the *kernType* parameter. The size of the Sobel kernel may be specified the parameter *apertureSize*. If this parameter is set to 3 - the function used 3x3 kernel, if it set to 5 - the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter *apertureSize* must be set to 3 if the Scharr kernel is used.



**CAUTION.** If the parameter *apertureSize* is set to 5 for operation with the Scharr kernel, the function returns error status.

Then, the function computes eigen values and vectors of the following matrix:

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix}.$$



The summation is performed over the full block with averaging over the blurring window with size *avgWindow*.

The image *eigenVV* has the following format. For every pixel of the source image it contains six floating-point values –  $\lambda_1$ ,  $\lambda_2$ ,  $x_1$ ,  $y_1$ ,  $x_2$ ,  $y_2$ . These values are defined as follows:

$\lambda_1$ ,  $\lambda_2$  Eigen values of the above matrix (not sorted by value).

$x_1$ ,  $y_1$  Coordinates of the normalized eigen vector corresponding to  $\lambda_1$ .

$x_2$ ,  $y_2$  Coordinates of the normalized eigen vector corresponding to  $\lambda_2$ .

In case of a singular matrix or when one eigen value is much smaller than the second one, all these six values are set to 0.

The function requires a temporary working buffer; its size should be computed previously by calling the function [ippiEigenValsVecsGetBufferSize](#).



---

**CAUTION.** The parameters *apertureSize* and *avgWindow* must be the same for both functions `ippiEigenValsVecsGetBufferSize` and `ippiEigenValsVecs`.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>pRoiSize</i> has a field with zero or negative value, or if <i>apertureSize</i> or <i>avgWindow</i> has an illegal value; or if <i>kernType</i> has wrong value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width*&lt;pixelSize&gt;</i> , or <i>eigStep</i> is less than <i>roiSize.width*sizeof(Ipp32f)*6</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4.

## MinEigenValGetBufferSize

*Calculates size of temporary buffer for the function `ippiMinEigenVal`.*

### Syntax

```
IppStatus ippiMinEigenValGetBufferSize_32f_C1R(IppiSize roiSize, int
    apertureSize, int avgWindow, int* pBufferSize);
IppStatus ippiMinEigenValGetBufferSize_8u32f_C1R(IppiSize roiSize, int
    apertureSize, int avgWindow, int* pBufferSize);
```

### Parameters

<i>roiSize</i>	Size of the source image ROI in pixels.
<i>apertureSize</i>	Size (in pixels) of the derivative operator used by the function, possible values are 3 or 5.
<i>avgWindow</i>	Size of the blurring window in pixels, possible values are 3 or 5.
<i>pBufSize</i>	Pointer to the variable that returns the size of the temporary buffer.

### Description

The function `ippiMinEigenValGetBufferSize` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function calculates the size of a temporary buffer to be used by the function [ippiMinEigenVal](#).



**CAUTION.** The parameters *apertureSize* and *avgWindow* must be the same for both functions `ippiMinEigenValGetBufferSize` and `ippiMinEigenVal`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiWidth</i> has a zero or negative value, or if <i>apertureSize</i> or <i>avgWindow</i> has an illegal value.

---

## MinEigenVal

*Calculates the minimal eigen value of image blocks for corner detection.*

---

### Syntax

```
IppStatus ippiMinEigenVal_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
    pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType
    kernType, int apertureSize, int avgWindow, Ipp8u* pBuffer);

IppStatus ippiMinEigenVal_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
    pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType
    kernType, int apertureSize, int avgWindow, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.				
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.				
<i>pMinEigenVal</i>	Pointer to the image that is filled with the minimal eigen values.				
<i>minValStep</i>	Distance in bytes between starts of consecutive lines in the output image.				
<i>roiSize</i>	Size of the source image ROI in pixels.				
<i>kernType</i>	Specifies the type of kernel used to compute derivatives, possible values are: <div><table><tr><td><code>ippKernelSobel</code></td><td>Sobel kernel 3x3 or 5x5;</td></tr><tr><td><code>ippKernelScharr</code></td><td>Scharr kernel 3x3.</td></tr></table></div>	<code>ippKernelSobel</code>	Sobel kernel 3x3 or 5x5;	<code>ippKernelScharr</code>	Scharr kernel 3x3.
<code>ippKernelSobel</code>	Sobel kernel 3x3 or 5x5;				
<code>ippKernelScharr</code>	Scharr kernel 3x3.				
<i>apertureSize</i>	Size of the derivative operator in pixels, possible values are 3 or 5.				
<i>avgWindow</i>	Size of the averaging window in pixels, possible values are 3 or 5.				
<i>pBuffer</i>	Pointer to the temporary buffer.				

### Description

The function `ippiMinEigenVal` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function takes a block around the pixel and computes the first derivatives  $D_x$  and  $D_y$ . This operation is performed for every pixel of the image using either

Sobel or Scharr kernel in accordance with the *kernType* parameter. The size of the Sobel kernel may be specified the parameter *apertureSize*. If this parameter is set to 3 - the function used 3x3 kernel, if it set to 5 - the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter *apertureSize* must be set to 3 if the Scharr kernel is used.



**CAUTION.** If the parameter *apertureSize* is set to 5 for operation with the Scharr kernel, the function returns error status.

Then, the function computes the minimal eigen value of the following matrix:

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix}.$$

The summation is performed over the full block with averaging over the blurring window with size *avgWindow*.

The function requires a temporary working buffer; its size should be computed previously by calling the function [ippiMinEigenValGetBufferSize](#).



**CAUTION.** The parameters *apertureSize* and *avgWindow* must be the same for both functions `ippiMinEigenValGetBufferSize` and `ippiMinEigenVal`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>pRoiSize</i> has a field with zero or negative value, or if <i>apertureSize</i> or <i>avgWindow</i> has an illegal value; or if <i>kernType</i> has wrong value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width*&lt;pixelSize&gt;</i> , or <i>eigenvvStep</i> is less than <i>roiSize.width*sizeof(Ipp32f)</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4.

## Distance Transform Functions

This section describes distance transform functions.

Distance transform is used for calculating the distance to an object. The input is an image with feature and non-feature pixels. The function labels every non-feature pixel in the output image with a distance to the closest feature pixel. Feature pixels are marked with zero.

Distance transform is used for a wide variety of subjects including skeleton finding and shape analysis.

---

### DistanceTransform

*Calculates distance to closest zero pixel for all non-zero pixels of source image.*

---

#### Syntax

##### Case 1: Not-in-place operations

```
IppStatus ippiDistanceTransform_3x3_<mod>(const Ipp8u* pSrc, int  
    srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,  
    Ipp32s* pMetrics);
```

```
IppStatus ippiDistanceTransform_5x5_<mod>(const Ipp8u* pSrc, int  
    srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize,  
    Ipp32s* pMetrics);
```

Supported values for *mod*:

8u\_C1R                      8u16u\_C1R

```
IppStatusippiDistanceTransform_3x3_8u32f_C1R(constIpp8u*pSrc,intsrcStep,  
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f* pMetrics);
```

```
IppStatusippiDistanceTransform_5x5_8u32f_C1R(constIpp8u*pSrc,intsrcStep,  
    Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f* pMetrics);
```

##### Case 2: In-place operations

```
IppStatus ippiDistanceTransform_3x3_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,  
    IppiSize roiSize, Ipp32s* pMetrics);
```

```
IppStatus ippiDistanceTransform_5x5_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep,
      IppiSize roiSize, Ipp32s* pMetrics);
```

Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the ROI in the destination distance image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the source and destination image ROI for in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>pMetrics</i>	Pointer to the array that specifies used metrics.

Description

The distance transform function `ippiDistanceTransform` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function approximates the actual distance from the closest zero pixel to each certain pixel with the sum of atomic distances from the fixed set. The set consists of two values for a 3×3 mask and three values for a 5×5 mask.

[Figure 14-2](#) shows the result of the distance transform of a 7×7 image with zero point in the center. This example corresponds to a 3×3 mask. Two numbers specify metrics in case of the 3×3 mask:

- distance between two pixels that share an edge,
- distance between the pixels that share a corner.

In this case the values are 1 and 1.5 correspondingly.

Figure 14-2 3x3 Mask

4.5	4	3.5	3	3.5	4	4.5
4	3	2.5	2	2.5	3	4

**Figure 14-2 3x3 Mask**

---

4.5	4	3.5	3	3.5	4	4.5
3.5	2.5	1.5	1	1.5	2.5	3.5
3	2	1	0	1	2	3
3.5	2.5	1.5	1	1.5	2.5	3.5
4	3	2.5	2	2.5	3	4
4.5	4	3.5	3	3.5	4	4.5

---

[Figure 14-3](#) shows the distance transform for the same image, but for a 5×5 mask.

For this mask yet another number is added to specify metrics - the additional distance, i.e., the distance between pixels corresponding to the chess knight move.

In this example, the additional distance is equal to 2.

**Figure 14-3 5x5 Mask**

---

4	3.5	3	3	3	3.5	4
3.5	3	2	2	2	3	3.5
3	2	1.5	1	1.5	2	3
3	2	1	0	1	2	3
3	2	1.5	1	1.5	2	3
3.5	3	2	2	2	3	3.5
4	3.5	3	3	3	3.5	4

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width*&lt;pixelSize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if step value is not divisible by 2 for 16u images, and by 4 for 32f images.

---

`ippStsCoeffErr` Indicates an error condition if at least one element of *pMetrics* array has zero or negative value.

---

## GetDistanceTransformMask

Returns an optimal mask for a given type of metrics and given mask size.

---

### Syntax

```
IppStatus ippGetDistanceTransformMask_<mod>(int kerSize, IppiNorm norm,
      Ipp<datatype>* pMetrics);
```

Supported values for *mod*:

32s                      32f

### Parameters

*kerSize* Specifies the mask size as follows: 3 for 3x3 mask, 5 for 5x5 mask.

*norm* Specifies the type of metrics. Possible values are:

`ippiNormInf(0)`     $L_{\infty}, \Delta = \max(|x_1 - x_2|, |y_1 - y_2|),$

`ippiNormL1(1)`     $L_1, \Delta = |x_1 - x_2| + |y_1 - y_2|,$

`ippiNormL2(2)`     $L_2, \Delta = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

*pMetrics* Pointer to the output array to store metrics parameters. The array contains the following number of elements:

2 for 3x3 mask,

3 for 5x5 mask.

### Description

The function `ippGetDistanceTransformMask` is declared in the `ippcv.h` file. This function fills up the output array with metrics parameters for the given type of metrics and size of mask.

The function returns the following results:

(1, 1)                       $L_{\infty}$ , 3x3 mask,

(1, 2)                       $L_1$ , 3x3 mask,



(2, 3)	$L_2$ , 3×3 mask, 32s data type,
(0.955, 1.3693)	$L_2$ , 3×3 mask, 32f datatype,
(1, 1, 2)	$L_\infty$ , 5×5 mask,
(1, 2, 3)	$L_1$ , 5×5 mask,
(4, 6, 9)	$L_2$ , 5×5 mask, 32s data type
(1.0, 1.4, 2.1969)	$L_2$ , 5×5 mask, 32f datatype.

$L_\infty$ , and  $L_1$  metrics for 5x5 mask are calculated using 3×3 mask,

For more information, see [[Borge86](#)].



---

**NOTE.** For compatibility with the previous versions of the library the earlier function `ippiGetDistanceTransformMask` replaced by the function `ippiGetDistanceTransformMask_32f` in the current version is also supported.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pMetrics</i> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>kerSize</i> has a wrong value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>kerSize</i> or <i>norm</i> has a wrong value.

---

## Image Gradients

---

### GradientColorToGray

*Converts a color gradient image to grayscale.*

---

#### Syntax

```
IppStatus ippiGradientColorToGray_<mod>(const Ipp<datatype>* pSrc, int  
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiNorm  
    norm);
```

Supported values for *mod* :

8u\_C3C1R      16u\_C3C1R      32f\_C1R

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.						
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.						
<i>pDst</i>	Pointer to the destination image ROI.						
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.						
<i>roiSize</i>	Size of the source and destination image ROI.						
<i>norm</i>	Type of norm to form the mask for dilation; following values are possible: <table><tr><td><i>ippiNormInf</i></td><td>Infinity norm.</td></tr><tr><td><i>ippiNormL1</i></td><td>L1 norm.</td></tr><tr><td><i>ippiNormL2</i></td><td>L2 norm.</td></tr></table>	<i>ippiNormInf</i>	Infinity norm.	<i>ippiNormL1</i>	L1 norm.	<i>ippiNormL2</i>	L2 norm.
<i>ippiNormInf</i>	Infinity norm.						
<i>ippiNormL1</i>	L1 norm.						
<i>ippiNormL2</i>	L2 norm.						

#### Description

The function `ippiGradientColorToGray` is declared in the `ippcv.h`. It operates with ROI (see Regions of Interest in Intel IPP).

This function creates the grayscale gradient image *pDst* from the source three-channel gradient image *pSrc*. The type of norm is specified by the parameter. Pixel values for destination image are computed for different type of norm in accordance with the following formula:

$$dst_{i,j} = \begin{cases} \max\{|src_{i,j,0}|, |src_{i,j,1}|, |src_{i,j,2}|\} & norm = ippiNormInf \\ |src_{i,j,0}| + |src_{i,j,1}| + |src_{i,j,2}| & norm = ippiNormL1 \\ \sqrt{src_{i,j,0}^2 + src_{i,j,1}^2 + src_{i,j,2}^2} & norm = ippiNormL2 \end{cases}$$

For integer flavors the result is scaled to the full range of of the destination data type.

### Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippiStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippiStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code> .
<code>ippiStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 2 for integer images, or by 4 for floating-point images.
<code>ippiStsBadArgErr</code>	Indicates an error condition if <code>norm</code> has an illegal value.

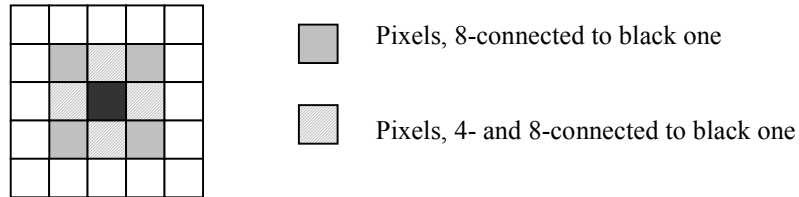
## Flood Fill Functions

This section describes functions performing flood filling of connected areas. *Flood filling* means that a group of connected pixels with close values is filled with, or is set to, a certain value. The flood filling process starts with a specified point (“seed”) and continues until it reaches the image ROI boundary or cannot find any new pixels to fill due to a large difference in pixel values. For every pixel filled, the functions analyze neighbor pixels:

- 4 neighbors (except diagonal neighbors); this kind of connectivity is called 4-connectivity and the corresponding function name includes `4Con`, or

- 8 neighbors (diagonal neighbors included); this kind of connectivity is called 8-connectivity and the corresponding function name includes 8Con.

**Figure 14-4**    **Pixels Connectivity Patterns**



These functions can be used for:

- segmenting a grayscale image into a set of uni-color areas,
- marking each connected component with individual color for bi-level images.

## FloodFillGetSize

*Calculates size of temporary buffer for flood filling operation.*

### Syntax

```
IppStatus ippiFloodFillGetSize(IppiSize roiSize, int* pBufSize);
```

### Parameters

*roiSize*                      Size of the source image ROI in pixels.  
*pBufSize*                     Pointer to the variable that returns the size of the temporary buffer.

### Description

The function `ippiFloodFillGetSize` is declared in the `ippcv.h` file. This function calculates the size of the temporary buffer to be used by the function [ippiFloodFill](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pBufSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## FloodFillGetSize\_Grad

*Calculates size of temporary buffer  
for the gradient flood filling.*

---

### Syntax

```
IppStatus ippIFloodFillGetSize_Grad(IppiSize roiSize, int* pBufSize);
```

### Parameters

<code>roiSize</code>	Size of the source image ROI in pixels.
<code>pBufSize</code>	Pointer to the variable that returns the size of the temporary buffer.

### Description

The function `ippIFloodFillGetSize_Grad` is declared in the `ippcv.h` file. This function calculates the size of the temporary buffer to be used by the function [ippIFloodFill\\_Grad](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pBufSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

## FloodFill

*Performs flood filling of connected area on an image.*

### Syntax

#### Case 1: Operations on one-channel data

```
IppStatus ippiFloodFill_4Con_<mod>(Ipp<datatype>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_8Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for *mod* :

8u\_C1IR            32f\_C1IR

#### Case 2: Operations on three-channel data

```
IppStatus ippiFloodFill_4Con_<mod>(Ipp<datatype>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_8Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for *mod* :

8u\_C3IR            32f\_C3IR

### Parameters

<i>pImage</i>	Pointer to the ROI in the source and destination image (for the in-place operation).
<i>imageStep</i>	Distance in bytes between starts of consecutive lines in the image buffer.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>seed</i>	Initial point.
<i>newVal</i>	Value to fill with for one-channel data.
<i>pNewVal</i>	Pointer to the vector containing values to fill with for three-channel data.

<i>pRegion</i>	Pointer to the connected components structure that stores information about the refilled area.
<i>pBuffer</i>	Pointer to the temporary buffer.

## Description

The function `ippiFloodFill` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs [flood filling](#) of the group of connected pixels whose pixel values are equal to the value in the *seed* point. Values of these pixel is set to the *newVal* value.

The function requires a temporary buffer whose size should be computed with the function [ippiFloodFillGetSize](#) beforehand.

The functions with the “\_4con” suffixes check 4-connected neighborhood of each pixel, that is, side neighbors. The functions with the “\_8con” suffixes check 8-connected neighborhood of each pixel, that is, side and corner neighbors. See [Figure 14-4](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>imageStep</i> is less than <i>pRoiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if the <i>seed</i> point is out of ROI.

## FloodFill\_Grad

*Performs gradient flood filling of connected area on an image.*

### Syntax

#### Case 1: Operations on one-channel data

```
IppStatus ippiFloodFill_Grad4Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>
    newVal, Ipp<datatype> minDelta, Ipp<datatype> maxDelta,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);

IppStatus ippiFloodFill_Grad8Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>
    newVal, Ipp<datatype> minDelta, Ipp<datatype> maxDelta,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for *mod* :

8u\_C1IR            32f\_C1IR

#### Case 2: Operations on three-channel data

```
IppStatus ippiFloodFill_Grad4Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>*
    pNewVal, Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);

IppStatus ippiFloodFill_Grad8Con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>*
    pNewVal, Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for *mod* :

8u\_C3IR            32f\_C3IR

### Parameters

<i>pImage</i>	Pointer to the ROI in the source and destination image (in-place operation).
<i>imageStep</i>	Distance in bytes between starts of consecutive lines in the image buffer.
<i>roiSize</i>	Size of the image ROI in pixels.



<i>seed</i>	Initial point.
<i>minDelta</i>	Minimum difference between neighbor pixels for one-channel data.
<i>maxDelta</i>	Maximum difference between neighbor pixels for one-channel data.
<i>newVal</i>	Value to fill with for one-channel data.
<i>pMinDelta</i>	Pointer to the minimum differences between neighbor pixels for three-channel images.
<i>pMaxDelta</i>	Pointer to the maximum differences between neighbor pixels for three-channel images.
<i>pNewVal</i>	Pointer to the vector containing values to fill with for three-channel data.
<i>pRegion</i>	Pointer to the connected components structure that stores information about the refilled area.
<i>pBuffer</i>	Pointer to the temporary buffer.

## Description

The function `ippiFloodFill_Grad` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs [flood filling](#) of the group of connected pixels in the *seed* pixel neighborhoods whose pixel values *v* satisfies the following conditions:

$$v_0 - d_{lw} \leq v \leq v_0 + d_{up},$$

where  $v_0$  is the value of at least one of the current pixel neighbors, which already belongs to the refilled area, and  $d_{lw}$ ,  $d_{up}$  are *minDelta*, *maxDelta*, respectively. Values of these pixel is set to the *newVal* value.

The function requires a temporary buffer whose size should be computed with the function [ippiFloodFillGetSize\\_Grad](#) beforehand.

The functions with the “\_4con” suffixes check 4-connected neighborhood of each pixel, i.e., side neighbors. The functions with the “\_8con” suffixes check 8-connected neighborhood of each pixel, i.e., side and corner neighbors. See [Figure 14-4](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .

---

<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>imageStep</i> is less than <i>pRoiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if the <i>seed</i> point is out of ROI.

---

## FloodFill\_Range

*Performs flood filling of pixels with values in the specified range in the connected area on an image.*

---

### Syntax

#### Case 1: Operations on one-channel data

```
IppStatus ippifloodfill_range4con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>
    newVal, Ipp<datatype> minDelta, Ipp<datatype> maxDelta,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);

IppStatus ippifloodfill_range8con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>
    newVal, Ipp<datatype> minDelta, Ipp<datatype> maxDelta,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for *mod* :

```
8u_C1IR      32f_C1IR
```

#### Case 2: Operations on three-channel data

```
IppStatus ippifloodfill_range4con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>*
    pNewVal, Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);

IppStatus ippifloodfill_range8con_<mod>(Ipp<DataType>* pImage,
    int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>*
    pNewVal, Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta,
    IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for *mod* :

8u\_C3IR                      32f\_C3IR

## Parameters

<i>pImage</i>	Pointer to the ROI in the source and destination image (in-place operation).
<i>imageStep</i>	Distance in bytes between starts of consecutive lines in the image buffer.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>seed</i>	Initial point.
<i>minDelta</i>	Minimum difference between neighbor pixels for one-channel data.
<i>maxDelta</i>	Maximum difference between neighbor pixels for one-channel data.
<i>newVal</i>	Value to fill with for one-channel data.
<i>pMinDelta</i>	Pointer to the minimum differences between neighbor pixels for three-channel images.
<i>pMaxDelta</i>	Pointer to the maximum differences between neighbor pixels for three-channel images.
<i>pNewVal</i>	Pointer to the vector containing values to fill with for three-channel data.
<i>pRegion</i>	Pointer to the connected components structure that stores information about the refilled area.
<i>pBuffer</i>	Pointer to the temporary buffer.

## Description

The function `ippiFloodFill_Grad` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs [flood filling](#) of the group of connected pixels in the *seed* pixel neighborhoods whose pixel values *v* satisfies the following conditions:

$$v_0 - d_{lw} \leq v \leq v_0 + d_{up},$$

where  $v_0$  is the pixel value of the *seed* point, and  $d_{lw}$ ,  $d_{up}$  are *minDelta*, *maxDelta*, respectively. Values of these pixel is set to the *newVal* value.

The function requires a temporary buffer whose size should be computed with the function [ippiFloodFillGetSize](#) beforehand.

The functions with the “\_4con” suffixes check 4-connected neighborhood of each pixel, i.e., side neighbors. The functions with the “\_8con” suffixes check 8-connected neighborhood of each pixel, i.e., side and corner neighbors. See [Figure 14-4](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>imageStep</code> is less than <code>pRoiSize.width * &lt;pixelsize&gt;</code> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if the <code>seed</code> point is out of ROI.

## Motion Analysis and Object Tracking

### Motion Template Functions

This section describes a motion templates function.

This function generates motion templates images to rapidly determine where, how, and in which direction the motion occurred. The algorithms are based on [\[Davis97\]](#), and [\[Davis99\]](#). The function operates on images that are the output of frame or background differencing, or other image segmentation operations. Thus, the input and output image types are all grayscale, that is, one color channel. The pixel types can be `8u`, `8s`, or `32f`.

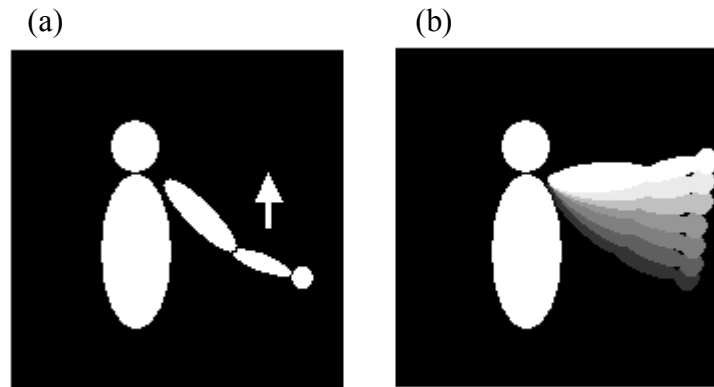
### Motion Representation

[Figure 14-5](#) (a) shows capturing a foreground silhouette of the moving object or person. As the person or object moves, copying the most recent foreground silhouette as the highest values in the motion history image creates a “layered history” of the resulting motion. Typically, this “highest value” is just a floating-point timestamp of time since the code has been running in milliseconds.

[Figure 14-5](#) (b) shows the result that may be called the *Motion History Image (MHI)*. The MHI in [Figure 14-5](#) represents how the motion took place. A pixel level or a time delta threshold, as appropriate, is set such that pixel values in the MHI that fall below that threshold are set to zero.

**Figure 14-5 Motion Image History**

---



The most recent motion has the highest value, earlier motions have decreasing values subject to a threshold below which the value is set to zero.

### Updating MHI Images

Generally, floating point images are used because system time differences, that is, time elapsing since the application was launched, are read in milliseconds to be further converted into a floating point number which is the value of the most recent silhouette. Then follows writing this current silhouette over the past silhouettes with subsequent thresholding away pixels that are too old to create the MHI.

---

## UpdateMotionHistory

*Updates motion history image using motion silhouette at given timestamp.*

---

### Syntax

```
IppStatus ippiUpdateMotionHistory_<mod>(const Ipp<srcDatatype>*
    pSilhouette, int silhStep, Ipp32f* pMhi, int mhiStep, IppiSize roiSize,
    Ipp32f timeStamp, Ipp32f mhiDuration);
```

Supported values for *mod* :

```
8u32f_C1IR      16u32f_C1IR      32f_C1IR
```

### Parameters

<i>pSilhouette</i>	Pointer to the silhouette image ROI that has non-zero values for those pixels where the motion occurs.
<i>roiSize</i>	Size of the image ROI in pixels.
<i>silhStep</i>	Distance in bytes between starts of consecutive lines in the silhouette image.
<i>pMhi</i>	Pointer to the motion history image which is both an input and output parameter.
<i>mhiStep</i>	Distance in bytes between starts of consecutive lines in the motion history image.
<i>timeStamp</i>	Timestamp in milliseconds.
<i>mhiDuration</i>	Threshold for MHI pixels. MHI motions older than this threshold are deleted.

### Description

The function `ippiUpdateMotionHistory` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function updates the motion history image. It sets MHI pixels to the current *timeStamp* value, if their values are non-zero.

The function deletes MHI pixels, if their values are less than the *mhiDuration* timestamp, that is, the pixels are “old.”

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>mhiStep</i> or <i>silhStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if the step value is not divisible by 2 for 16u images, and by 4 for 32f images.
<code>ippStsOutOfRangeErr</code>	Indicates an error when <i>mhiDuration</i> is negative.

## Optical Flow

This section describes the functions that calculate the optical flow using the pyramidal Lucas-Kanade algorithm [[Bouguet99](#)].

The optical flow between two images is generally defined as an apparent motion of image brightness. Let  $I(x, y, t)$  be the image brightness that changes in time to provide an image sequence.

Optical flow coordinates  $u = \frac{\partial x}{\partial t}$ ,  $v = \frac{\partial y}{\partial t}$  can be found from so called *optical flow constraint equation*:

$$-\frac{\partial I}{\partial t} = \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

The Lucas-Kanade algorithm assumes that the group of adjacent pixels has the same velocity and finds the approximate solution of the above equation using the least square method.

---

## OpticalFlowPyrLKInitAlloc

*Allocates memory and initializes a structure for optical flow calculation.*

---

### Syntax

```
IppStatus ippiOpticalFlowPyrLKInitAlloc_<mod>(IppiOptFlowPyrLK_<mod>** ppState,
        IppiSize roiSize, int winSize, IppHintAlgorithm hint);
```

Supported values for *mod* :

8u\_C1R                  16u\_C1R                  32f\_C1R

### Parameters

<i>ppState</i>	Pointer to the pointer to the optical flow structure being initialized.
<i>roiSize</i>	Size of the source image (zero level of the pyramid) ROI in pixels.
<i>winSize</i>	Size of the search window of each pyramid level.
<i>hint</i>	Option to select the algorithmic implementation of the transform function

### Description

The function `ippiOpticalFlowPyrLKInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function allocates memory and initializes a structure *pState* for calculation the optical flow between two images using the pyramidal Lucas-Kanade algorithm in the centered window of size *winSize* \* *winSize*. Computation algorithm is specified by the *hint* argument. This structure is used by the function [ippiOpticalFlowPyrLK](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>ppState</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value or if <i>winSize</i> is equal to or less than 0.
<code>ippStsMemAllocErr</code>	Memory allocation error.



---

## OpticalFlowPyrLKFree

*Frees memory allocated for the optical flow structure.*

---

### Syntax

```
IppStatus ippiOpticalFlowPyrLKFree_<mod>(IppiOptFlowPyrLK_<mod>* pState);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
--------	---------	---------

### Parameters

<i>pState</i>	Pointer to the optical flow structure.
---------------	--

### Description

The function `ippiOpticalFlowPyrLKFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function [ippiOpticalFlowPyrLKInitAlloc](#) for the optical flow structure.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> is NULL.

---

## OpticalFlowPyrLK

*Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.*

---

### Syntax

```
IppStatus ippiOpticalFlowPyrLK_<mod>(IppiPyramid* pPyr1, IppiPyramid* pPyr2,  
    const IppiPoint2D32f* pPrev, IppiPoint_32f* pNext, Ipp8s* pStatus, Ipp32f*  
    pError, int numFeat, int winSize, int maxLev, int maxIter, Ipp32f threshold,  
    IppiOptFlowPyrLK_<mod>* pState);
```

Supported values for *mod* :

8u\_C1R                  16u\_C1R                  32f\_C1R

## Parameters

<i>pPyr1</i>	Pointer to the ROI in the first image pyramid structure.
<i>pPyr2</i>	Pointer to the ROI in the second image pyramid structure.
<i>pPrev</i>	Pointer to the array of initial coordinates of the feature points.
<i>pNext</i>	Pointer to the array of new coordinates of feature point; as input it contains hints for new coordinates.
<i>pStatus</i>	Pointer to the array of result indicators.
<i>pError</i>	Pointer to the array of differences between neighborhoods of old and new point positions.
<i>numFeat</i>	Number of feature points.
<i>winSize</i>	Size of the square search window.
<i>maxLev</i>	Pyramid level to start the operation.
<i>maxIter</i>	Maximum number of algorithm iterations for each pyramid level.
<i>threshold</i>	Threshold value.
<i>pState</i>	Pointer to the pyramidal optical flow structure.

## Description

The function `ippiOpticalFlowPyrLK` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function implements the iterative version of the Lucas-Kanade algorithms [[Bouguet99](#)]. It computes with sub-pixel accuracy new coordinates of the *numFeat* feature points of two images at time  $t$  and  $t+dt$ . Their initial coordinates are places in the *pPrev* array. Computed values of new coordinates of the feature points are stored in the array *pNext*, that initially contains estimations of them (or hints), for example, based on the flow values for the previous image in sequence. If there are not such hints, the *pNext* array contains the same initial coordinates as the *pPrev* array.

The images are presented by the pyramid structures *pPyr1* and *pPyr2* respectively (see description of the [ippiPyramidInitAlloc](#) function for more details). These structures should be initialized by calling the function [ippiPyramidInitAlloc](#) beforehand. Furthermore the function uses the pyramidal optical flow structure *pState* that also should be previously initialized by the function [ippiOpticalFlowPyrLKInitAlloc](#).

The function starts operation on the highest pyramid level (smallest image) that is specified by the *maxLev* parameter in the centered search window whose size *winSize* could not exceed the corresponding value *winSize* that is specified in the function [ippiOpticalFlowPyrLKInitAlloc](#). The operation for *i*-th feature point on the given pyramid level finishes if:

- new position of the point is found ( $\sqrt{dx^2 + dy^2} < threshold$ ),
- specified number of iteration *maxIter* is performed,
- the intersection between the pyramid layer and the search window became empty.

In first two cases for non-zero levels the new position coordinates are scaled to the next pyramid level and the operation continues on the next level.

For zero level or for third case the operation stops, the number of the corresponding level is written to the *pStatus[i]* element, the new coordinates are scaled to zero level and are written to *pNext[i]*. The square root of the average squared difference between neighborhoods of old and new positions is written to *pError[i]*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>numFeat</i> or <i>winSize</i> has zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>maxLev</i> or <i>threshold</i> has negative value, or <i>maxIter</i> has zero or negative value.

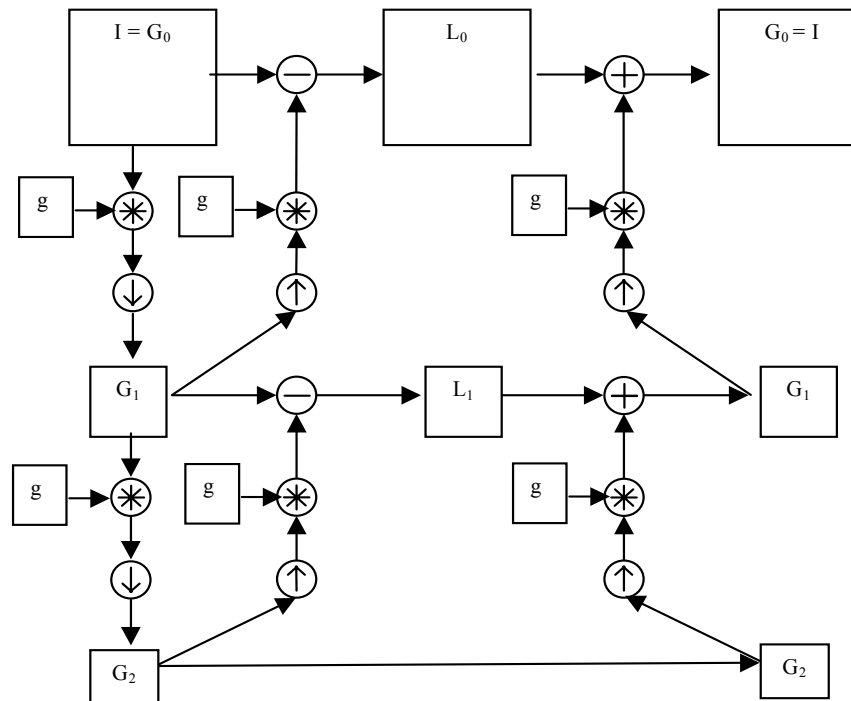
## Pyramids Functions

The functions described in this section generate and reconstruct Gaussian and Laplacian image pyramids.

[Figure 14-6](#) shows the basics of creating Gaussian or Laplacian pyramids. The original image  $G_0$  is convolved with a Gaussian, then down-sampled to get the reduced image  $G_1$ . This process can be continued as far as desired or until the image size is one pixel.

The Laplacian pyramid can be built from a Gaussian pyramid as follows: the Laplacian level  $k$ , that is,  $L_k$  can be built by up-sampling the lower level image  $G_{k+1}$ . Convolving the image with a Gaussian kernel  $g$  interpolates the pixels “missing” after up-sampling. The resulting image is subtracted from the Gaussian smoothed image  $G_k$ . To rebuild the original image, the process is reversed as [Figure 14-6](#) shows.

**Figure 14-6 Three-level Gaussian and Laplacian Pyramid**



The Gaussian image pyramid on the left is used to create the Laplacian pyramid in the center, which is used to reconstruct the Gaussian pyramid and original image on the right. In [Figure 14-6](#),  $I$  is the original image,  $G_k$  is the Gaussian image,  $L_k$  is the Laplacian image, subscripts  $k$  denote level of the pyramid,  $g$  is a Gaussian kernel used to convolve the image *before* down-sampling or *after* up-sampling.

The Intel IPP implement only Gaussian pyramids. To build Laplacian pyramids, the following general rule for Intel IPP pyramids functions is to be followed:

$$L_i = G_i - \text{PyrUp}(\text{PyrDown}(G_i)),$$

where  $L_i$  is an image on the  $i$ -th level in the Laplacian pyramid,  $G_i$  is an image on the  $i$ -th level in the Gaussian pyramid.

---

## PyrDownGetBufSize

*Computes the size of temporary buffer for the function `ippiPyrDown`.*

---

### Syntax

```
IppStatusippiPyrDownGetBufSize_Gauss5x5(int roiWidth, IppDataTypedataType,  
int channels, int* pBufSize);
```

### Parameters

<i>roiWidth</i>	Width of the source image ROI in pixels.
<i>dataType</i>	Data type of the source image.
<i>channels</i>	Number of channels.
<i>pBufSize</i>	Pointer to the computed size of the temporary buffer.

### Description

The function `ippiPyrDownGetBufSize` is declared in the `ippcv.h` file. This function calculates the size of the temporary buffer to be used by the function [ippiPyrDown](#). The buffer of the calculated size can be used to process smaller images.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
--------------------------	--

---

<code>ippStsNullPtr</code>	Indicates an error condition if the <code>pBufSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the value of <code>roiWidth</code> is zero or negative.
<code>ippStsNumChannelsErr</code>	Indicates an error condition if <code>channels</code> is not 1 or 3.
<code>ippStsDataTypeErr</code>	Indicates an error condition if <code>dataType</code> is not <code>Ipp8u</code> , <code>Ipp8s</code> , or <code>Ipp32f</code> .

---

## PyrUpGetBufSize

*Calculates size of temporary buffer  
for the function `ippiPyrUp`.*

---

### Syntax

```
IppStatus ippiPyrUpGetBufSize_Gauss5x5(int roiWidth, IppDataType dataType,
    int channels, int* pBufSize);
```

### Parameters

<code>roiWidth</code>	Width of the source image ROI in pixels.
<code>dataType</code>	Data type of the source image.
<code>channels</code>	Number of channels.
<code>pBufSize</code>	Pointer to the computed size of the temporary buffer.

### Description

The function `ippiPyrUpGetBufSize` is declared in the `ippcv.h` file. This function calculates the size of the temporary buffer to be used by the function [ippiPyrUp](#). The buffer of the calculated size can be used to process smaller images.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtr</code>	Indicates an error condition if the <code>pBufSize</code> pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the value of <code>roiWidth</code> is zero or negative.
<code>ippStsNumChannelsErr</code>	Indicates an error condition if <code>channels</code> is not 1 or 3.

`ippStsDataTypeErr` Indicates an error condition if *dataType* is not `Ipp8u`, `Ipp8s`, or `Ipp32f`.

---

## PyrDown

*Applies the Gaussian to image and then performs down-sampling.*

---

### Syntax

```
IppStatus ippiPyrDown_Gauss5x5_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>8s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>8s_C3R</code>	<code>32f_C3R</code>

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source image ROI in pixels.
<i>pBuffer</i>	Pointer to the temporary buffer.

### Description

The function `ippiPyrDown` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the 5×5 Gaussian to the source image *pSrc* and then down-samples it, that is, removes odd rows and columns from the image. The size of the destination image is  $(roiSize.height+1)/2 * (roiSize.width+1)/2$ . The following Gaussian mask is used:

$$\frac{1}{16} \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \times \frac{1}{16} [1 \ 4 \ 6 \ 4 \ 1] = \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

The function uses the temporary buffer *pBuffer* - its size should be computed using the function [ippiPyrDownGetBufSize](#) beforehand.

### Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than <i>(roiSize.width * &lt;pixelSize&gt;)/2</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if steps for floating-point images are not divisible by 4.

---

## PyrUp

*Up-samples image and then applies the Gaussian.*

---

### Syntax

```
ippStatus ippiPyrUp_Gauss5x5_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for *mod* :

<code>8u_C1R</code>	<code>8s_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>8s_C3R</code>	<code>32f_C3R</code>



## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source image ROI in pixels.
<i>pBuffer</i>	Pointer to the temporary buffer.

## Description

The function `ippiPyrUp` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function up-samples the source image *pSrc*, that is, inserts odd zero rows and columns, and then applies the 5×5 Gaussian multiplied by 4 to it. The following mask is used:

$$4 \times \frac{1}{16} \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{bmatrix} \times \frac{1}{16} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \frac{1}{64} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

The function uses the temporary buffer *pBuffer* - its size should be computed using the function [ippiPyrUpGetBufSize](#) beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error. Any other value indicates an error or a warning.
<code>ippStsNullPtr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than <i>2 * roiSize.width * &lt;pixelSize&gt;</i> .

`ippStsNotEvenStepErr` Indicates an error condition if when steps for floating-point images are not divisible by 4.

## Universal Pyramids

The functions described in this section operate with universal image pyramids. These pyramids use separable symmetric kernel (not only Gaussian type) and downsampling/upsampling with arbitrary factor (not only 2). The next pyramid layer can be built for an image of an arbitrary size. These pyramids are used in some computer vision algorithms, for example, in [optical flow](#) calculations.

---

## PyramidInitAlloc

*Allocates memory and initializes a pyramid structure.*

---

### Syntax

```
IppStatus ippPyramidInitAlloc(IppiPyramid** ppPyr, int level, IppiSize roiSize,
    Ipp32f rate);
```

### Parameters

<code>ppPyr</code>	Pointer to the pointer to the pyramid structure.
<code>level</code>	Maximum number of pyramid levels.
<code>roiSize</code>	Size of zero level image ROI in pixels.
<code>rate</code>	Ratio between neighbouring levels ( $1 < rate \leq 10$ ).

### Description

The function `ippPyramidInitAlloc` is declared in the `ippcv.h` file. This function allocates memory and initializes the structure for the pyramid with `level+1` levels. This structure is used by the [ippiOpticalFlowPyrLK](#) function for optical flow calculations.

The `IppiPyramid` structure contains the following fields:

<code>pImage</code>	Pointer to the array of <code>(level+1)</code> layer images.
<code>pStep</code>	Pointer to the array of <code>(level+1)</code> image row step values.
<code>pRoi</code>	Pointer to the array of <code>(level+1)</code> layer image ROIs.

<i>pRate</i>	Pointer to the array of $(level+1)$ ratios of $i$ -th levels to the zero level ( $rate^i$ ).
<i>pState</i>	Pointer to the structure to compute the next pyramid layer.
<i>level</i>	Number of levels in the pyramid.

The `ippiPyramidInitAlloc` function fills *pRoi* and *pRate* arrays and *level* field. The value of *level* is equal to the minimum of the input value of the *level* parameter and the maximum possible number of layers of the pyramid with given *rate* and zero level size.

Other fields should be specified by the user. The pyramid layer structure *pState* should be initialized by the functions [ippiPyramidLayerDownInitAlloc](#) or [ippiPyramidLayerUpInitAlloc](#). Pyramid layer images can be obtained in many different ways, for example using the Intel IPP functions [ippiPyramidLayerDown](#) and [ippiPyramidLayerUp](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>ppPyr</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>level</i> is equal to or less than 0, or if <i>rate</i> is out of the range.
<code>ippStsMemAllocErr</code>	Memory allocation error.

---

## PyramidFree

*Frees memory allocated for the pyramid structure.*

---

### Syntax

```
IppStatus ippiPyramidFree(IppiPyramid* pPyr);
```

### Parameters

<i>pPyr</i>	Pointer to the pyramid structure.
-------------	-----------------------------------

## Description

The function `ippiPyramidFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function [ippiPyramidInitAlloc](#) for the pyramid structure `pPyr`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pPyr</code> pointer is NULL.

---

## PyramidLayerDownInitAlloc

*Allocates memory and initializes a structure for creating a lower pyramid layer.*

---

## Syntax

### Case 1: Operation on integer data

```
IppStatus ippiPyramidLayerDownInitAlloc_<mod>(IppiPyramidDownState_<mod>**
    ppState, IppiSize srcRoiSize, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode);
```

Supported values for `mod`:

<code>8u_C1R</code>	<code>16u_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>

### Case 2: Operation on floating point data

```
IppStatus ippiPyramidLayerDownInitAlloc_<mod>(IppiPyramidDownState_<mod>**
    ppState, IppiSize srcRoiSize, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode);
```

Supported values for `mod`:

<code>32f_C1R</code>
<code>32f_C3R</code>

## Parameters

<code>ppState</code>	Pointer to the pointer to the pyramid layer structure.
<code>srcRoiSize</code>	Maximal size of source image ROI in pixels.
<code>rate</code>	Ratio between neighbouring levels ( $1 < rate \leq 10$ ).

<i>pKernel</i>	Pointer to the symmetric separable convolution kernel.
<i>kerSize</i>	Size of the convolution kernel, should be odd and greater than 2.
<i>mode</i>	Specifies the type of interpolation for image downsampling; should be set to <code>IPPI_INTER_LINEAR</code> to perform bilinear interpolation.

## Description

The function `ippiPyramidLayerDownInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory and initializes the structure *pState* to build a lower pyramid layer. This structure is used by the function [ippiPyramidLayerDown](#) and can be applied to process images with size not greater than *srcRoiSize*.

Generally the specified kernel *pKernel* should be symmetric. However if it is not symmetric, the function builds the symmetric kernel using its first half, and returns the warning.

For integer rates downsampling is performed by discarding rows and columns that are not multiples of the rate value. For non-integer rate bilinear interpolation is used (see appendix B [“Linear Interpolation”](#) for more information).

The symmetric separable kernel of odd size can be not Gaussian. If the sum of kernel elements is not equal to zero, then the kernel is normalized.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> has a field with zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if one of the parameters <i>rate</i> , <i>kerSize</i> , <i>mode</i> has wrong value.
<code>ippStsMemAllocErr</code>	Memory allocation error.
<code>ippStsSymKernelExpected</code>	Indicates a warning if the specified kernel is not symmetric.

---

## PyramidLayerDownFree

*Frees memory allocated for the lower pyramid layer structure.*

---

### Syntax

```
IppStatus ippPyramidLayerDownFree_<mod>(IppiPyramidDownState_<mod>* pState);
```

Supported values for *mod* :

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

### Parameters

*pState*                      Pointer to the pyramid layer structure .

### Description

The function `ippPyramidLayerDownFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function [ippPyramidLayerDownInitAlloc](#) for the lower pyramid layer structure *pState*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> is NULL.

---

## PyramidLayerUpInitAlloc

*Allocates memory and initializes a structure for creating an upper pyramid layer.*

---

### Syntax

#### Case 1: Operation on integer data

```
IppStatus ippPyramidLayerUpInitAlloc_<mod>(IppiPyramidUpState_<mod>** ppState,  
IppiSize dstRoiSize, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode);
```

Supported values for *mod* :

8u_C1R	16u_C1R
8u_C3R	16u_C3R

### Case 2: Operation on floating point data

```
IppStatus ippiPyramidLayerUpInitAlloc_<mod>(IppiPyramidUpState_<mod>** ppState,  
IppiSize dstRoiSize, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode);
```

Supported values for *mod* :

32f_C1R
32f_C3R

### Parameters

<i>pState</i>	Pointer to the pointer to the pyramid layer structure.
<i>dstRoiSize</i>	Maximal size of the destination image ROI in pixels.
<i>rate</i>	Ratio between neighbouring levels ( $1 < rate \leq 10$ ).
<i>pKernel</i>	Pointer to the symmetric separable convolution kernel.
<i>kerSize</i>	Size of the convolution kernel, should be odd and greater than 2.
<i>mode</i>	Specifies the type of interpolation for image upsampling; should be set to <code>IPPI_INTER_LINEAR</code> to perform bilinear interpolation.

### Description

The function `ippiPyramidLayerUpInitAlloc` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function allocates memory and initializes the structure *pState* to build an upper pyramid layer. This structure is used by the function [ippiPyramidLayerUp](#) and can be applied to process images with size not greater than *dstRoiSize*.

Generally the specified kernel *pKernel* should be symmetric. However if it is not symmetric, the function builds the symmetric kernel using its first half, and returns the warning.

For integer rates upsampling is performed by inserting zero rows and columns that are not multiples of the rate value. For non-integer rate bilinear interpolation is used to calculate kernel coefficients for pixels with non-integer coordinates.

The symmetric separable kernel of odd size can be not Gaussian. If the sum of kernel elements is not equal to zero, then the kernel is normalized.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstRoiSize</code> has a field with zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if one of the parameters <code>rate</code> , <code>kerSize</code> , <code>mode</code> has wrong value, or the sum of the kernel elements is not positive.
<code>ippStsMemAllocErr</code>	Memory allocation error.
<code>ippStsSymKernelExpected</code>	Indicates a warning if the specified kernel is not symmetric.

---

## PyramidLayerUpFree

*Frees memory allocated for the upper pyramid layer structure.*

---

### Syntax

```
IppStatus ippPyramidLayerUpFree_<mod>(IppiPyramidUpState_<mod>* pState);
```

Supported values for `mod` :

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>32f_C3R</code>

### Parameters

`pState`                      Pointer to the pyramid layer structure.

### Description

The function `ippPyramidLayerUpFree` is declared in the `ippcv.h` file. This function frees memory allocated by the function [ippPyramidLayerUpInitAlloc](#) for the upper pyramid layer structure `pState`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------



`ippStsNullPtrErr` Indicates an error if `pState` is NULL.

---

## GetPyramidDownROI

*Computes the size of the lower pyramid layer.*

---

### Syntax

```
IppStatus ippGetPyramidDownROI(IppiSize srcRoiSize, IppiSize* pDstRoiSize,  
                                Ipp32f rate);
```

### Parameters

<code>srcRoiSize</code>	Size of the source pyramid layer ROI in pixels.
<code>pDstRoiSize</code>	Pointer to the size of the destination (lower) pyramid layer ROI in pixels.
<code>rate</code>	Ratio between source and destination layers ( $1 < rate \leq 10$ ).

### Description

The function `ippGetPyramidDownROI` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes the size of the lower pyramid layer `pDstRoiSize` for a source layer of a given size `srcRoiSize` and specified size ratio `rate` between them in accordance with the following formulas:

```
pDstRoiSize.width = max(1, min(⌈srcRoiSize.width/rate⌉, srcRoiSize.width-1))  
pDstRoiSize.height = max(1, min(⌈srcRoiSize.height/rate⌉, srcRoiSize.height-1))
```



---

**NOTE.** Since for the non-integer `rate` results depend on the computational precision, it is strongly recommended to use this function in computations.

---

### Return Values

`ippStsNoErr` Indicates no error.

---

<code>ippStsNullPtrErr</code>	Indicates an error if <code>pDstRoiSize</code> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoiSize</code> has a field with zero or negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <code>rate</code> is out of the range.

---

## GetPyramidUpROI

*Computes the size of the upper pyramid layer.*

---

### Syntax

```
IppStatus ippGetPyramidUpROI(IppiSize srcRoiSize, IppiSize* pDstRoiSizeMin,
                             IppiSize* pDstRoiSizeMax, Ipp32f rate);
```

### Parameters

<code>srcRoiSize</code>	Size of the source pyramid layer ROI in pixels.
<code>pDstRoiSizeMin</code>	Pointer to the minimal size of the destination (upper) pyramid layer ROI in pixels.
<code>pDstRoiSizeMax</code>	Pointer to the maximal size of the destination (upper) pyramid layer ROI in pixels.
<code>rate</code>	Ratio between source and destination layers ( $1 < rate \leq 10$ ).

### Description

The function `ippGetPyramidUpROI` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function computes possible sizes of the upper pyramid layer `pDstRoiSizeMin` and `pDstRoiSizeMax` for a source layer of a given size `srcRoiSize` and specified size ratio `rate` between them in accordance with the following formulas:

maximum size `pDstRoiSizeMax`

```
pDstRoiMax.width = max(srcRoiSize.width + 1, ⌊srcRoiSize.width · rate⌋)
pDstRoiMax.height = max(srcRoiSize.height + 1, ⌊srcRoiSize.height · rate⌋)
```

minimum size *pDstRoiSizeMin*

if the width and height of the source layer ROI is greater than 1:

```
pDstRoiMin.width = max(srcRoiSize.width + 1, ⌊(srcRoiSize.width - 1) · rate⌋)
pDstRoiMin.height = max(srcRoiSize.height + 1, ⌊(srcRoiSize.height - 1) · rate⌋)
```

if the width and height of the source layer ROI is equal to 1:

```
pDstRoiMin.width = 1
pDstRoiMin.height = 1
```




---

**NOTE.** Since for the non-integer *rate* results depend on the computational precision, it is strongly recommended to use this function in computations.

---

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error if one of the specified pointers is NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>srcRoiSize</i> has a field with zero or negative value.
<i>ippStsBadArgErr</i>	Indicates an error condition if <i>rate</i> is out of the range.

---

## PyramidLayerDown

*Creates a lower pyramid layer.*

---

### Syntax

```
IppStatus ippiPyramidLayerDown_<mod>(const Ipp<datatype>* pSrc, int
    srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep,
    IppiSize dstRoiSize, IppiPyramidDownState_<mod>* pState);
```

Supported values for *mod*:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

## Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of the source image ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of the destination image ROI in pixels.
<i>pState</i>	Pointer to the pyramid layer structure.

## Description

The function `ippiPyramidLayerDown` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates a lower pyramid layer *pDst* from the source image *pSrc*, that is, it applies the convolution kernel to the source image using the mirror border and then performs downsampling. Before calling `ippiPyramidLayerDown`, the pyramid layer structure *pState* should be initialized by calling the function [ippiPyramidLayerDownInitAlloc](#) and the kernel and the downsampling ratio should be specified beforehand.

The function can process images with *srcRoiSize* not greater than *srcRoiSize* parameter specified in the function [ippiPyramidLayerDownInitAlloc](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>srcRoiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than <i>dstRoiSize.width</i> * <i>&lt;pixelSize&gt;</i> .

<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error condition if <code>pState-&gt;rate</code> has wrong value.

---

## PyramidLayerUp

*Creates an upper pyramid layer.*

---

### Syntax

```
IppStatus ippPyramidLayerUp_<mod>(const Ipp<datatype>* pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
    IppiPyramidUpState_<mod>* pState);
```

Supported values for *mod*:

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>32f_C3R</code>

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcRoiSize</i>	Size of source image ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstRoiSize</i>	Size of destination image ROI in pixels.
<i>pState</i>	The pointer to the pyramid layer structure.

### Description

The function `ippPyramidLayerUp` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates an upper pyramid layer *pDst* from the source image *pSrc*, that is, it performs upsampling of the source image and then applies the convolution kernel using the mirror border. Before calling `ippiPyramidupUp` function, the pyramid layer structure *pState* should be initialized by calling the function [ippiPyramidLayerUpInitAlloc](#) and the kernel and the upsampling ratio should be specified beforehand.

The function can process images with *srcRoiSize* not greater than *srcRoiSize* parameter specified in the function [ippiPyramidLayerUpInitAlloc](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> or <i>dstRoiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>srcRoiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than <i>dstRoiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>pState-&gt;rate</i> has wrong value.

## Example of Using General Pyramid Functions

The following [Example 14-1](#) shows how different general pyramids functions can be used to create the Gaussian and Laplacian pyramids:

### Example 14-1 Building Gaussian and Laplacian Pyramids

---

```
void UsePyramids(Ipp32f *pSrc, IppiSize srcRoi, int srcStep, Ipp32f *pKernel,
                int kerSize) {
    IppiPyramid *gPyr;    // pointer to Gaussian pyramid structure
    IppiPyramid *lPyr;    // pointer to Laplacian pyramid structure

    // allocate pyramid structures
    ippiPyramidInitAlloc (&gPyr, 1000, srcRoi, rate);
    ippiPyramidInitAlloc (&lPyr, 1000, srcRoi, rate);
    {
        IppiPyramidDownState_32f_C1R **gState =
            (IppiPyramidDownState_32f_C1R*)&(gPyr->pState);
        IppiPyramidUpState_32f_C1R **lState =
            (IppiPyramidUpState_32f_C1R*) &(lPyr->pState);
        Ipp32f **gImage = (Ipp32f**) (gPyr->pImage);
        Ipp32f **lImage = (Ipp32f**) (lPyr->pImage);
        IppiSize *pRoi    = dPyr->pRoi;
        int *gStep = gPyr->pStep,
        int *lStep = lPyr->pStep;
        int level = gPyr->level;
        Ipp32f *ptr;
        int step;

        // allocate structures to calculate pyramid layers
        ippiPyramidLayerDownInitAlloc_32f_C1R(gState, srcRoi, rate, pKernel,
                                              kerSize, mode);
        ippiPyramidLayerUpInitAlloc_32f_C1R (lState, srcRoi, rate, pKernel,
                                              kerSize, mode);
    // build Gaussian pyramid with level+1 layers
    gImage[0] = pSrc;
    gStep[0]  = srcStep;
    for (i=1; i<=level; i++) {
        gImage[i] = ippiMalloc_32f_C1(pRoi[i].width,pRoi[i].height,gStep+i);
        ippiPyramidLayerDown_32f_C1R(gImage[i-1], gStep[i-1], pRoi[i-1],
                                     gImage[i], gStep[i], pRoi[i], *gState);
    }
}
```

---

**Example 14-1 Building Gaussian and Laplacian Pyramids (continued)**


---

```

//      build Laplacian pyramid with level layers
ptr = ippiMalloc_32f_C1(srcRoi.width,srcRoi.height,&step);
for (i=level-1; i>=0; i--) {
    lImage[i] = ippiMalloc_32f_C1(pRoi[i].width,pRoi[i].height,gStep+i);
    ippiPyramidLayerUp_32f_C1R(gImage[i+1], gStep[i+1], pRoi[i+1],
                               ptr, step, pRoi[i], *lState);
    ippiSub_32f_C1R(gImage[i], gStep[i], ptr, step,
                   lImage[i], lStep[i], pRoi[i]);
}
ippiFree(ptr);

ippiPyramidLayerDownFree_32f_C1R(*gStep);
ippiPyramidLayerUpFree_32f_C1R  (*lStep);

//      use Gaussian and Laplacian pyramids

//      free allocated images
for (i=1; i<=level; i++) {
    ippiFree(gImage[i]);
    ippiFree(lImage[i-1]);
}
// free pyramid structures
ippiPyramidFree (gPyr);
ippiPyramidFree (lPyr);
}

```

---



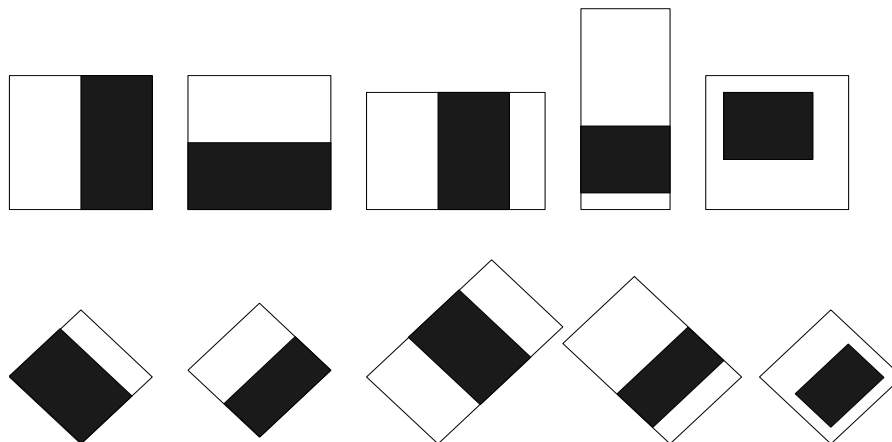
## Pattern Recognition

### Object Detection Using Haar-like Features

The object detector described in [Viola99] and [Leinhart02] is based on Haar classifiers. Each classifier uses  $k$  rectangular areas (Haar features) to make decision if the region of the image looks like the predefined image or not. [Figure 14-7](#) shows different types of Haar features.

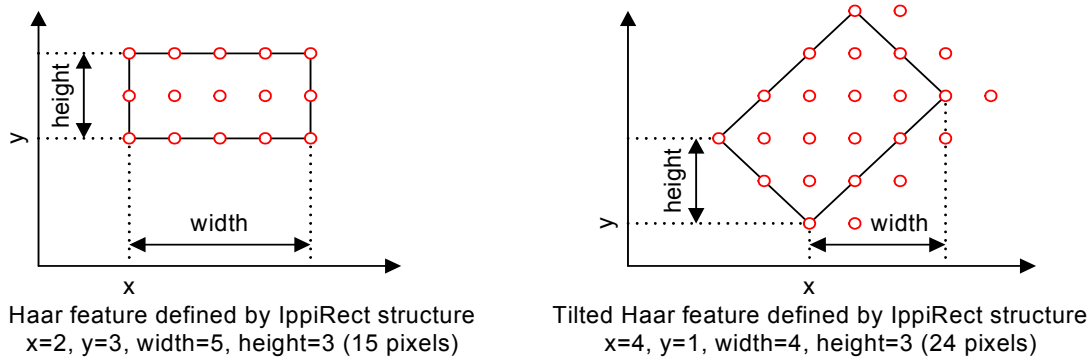
**Figure 14-7** Types of Haar Features

---



In the Intel IPP Haar features are represented using `IppRect` structure. [Figure 14-8](#) shows how it can be done for common and tilted features.

Figure 14-8 Representing Haar Features



When the classifier  $K_i$  is applied to the pixel  $(i, j)$  of the image  $A$ , it yields the value  $val1(t)$  if

$$\sum_{i=1}^k \left( w_1 \cdot \sum_{u=i+R_1y}^{R_1y+R_1height-1} \sum_{v=j+R_1yx}^{R_1y+R_1width-1} A_{uv} \right) < norm(i, j) \cdot threshold(t)$$

and  $val2(t)$  otherwise.

Here  $w_1$  is a feature weight,  $norm(i, j)$  is the norm factor (generally the standard deviation on the rectangle containing all features),  $threshold(t)$ ,  $val1(t)$  and  $val2(t)$  are parameters of the classifier. For fast computation the integral representation of an image is used. Haar classifiers are organized in sequences called *stages* (*classification stages*). The stage value is the sum of its classifier values. During feature detecting stages are consequently applied to the region of the image until the stage value becomes less than the threshold value or all stages are passed.

The use of the Intel IPP pattern recognition functions is demonstrated in the face detecting sample. See *Intel IPP JPEG Computer Vision Samples* downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>.

## HaarClassifierInitAlloc

*Allocates memory and initializes the structure for standard Haar classifiers.*

---

### Syntax

```
IppStatus ippiHaarClassifierInitAlloc_32f(IppiHaarClassifier_32f** pState,  
    const IppiRect* pFeature, const Ipp32f* pWeight, const Ipp32f* pThreshold,  
    const Ipp32f* pVal1, const Ipp32f* pVal2, const int* pNum, int length);  
IppStatus ippiHaarClassifierInitAlloc_32s(IppiHaarClassifier_32s** pState,  
    const IppiRect* pFeature, const Ipp32s* pWeight, const Ipp32s* pThreshold,  
    const Ipp32s* pVal1, const Ipp32s* pVal2, const int* pNum, int length);
```

### Parameters

<i>ppState</i>	Pointer to the pointer to the Haar classifier structure.
<i>pFeature</i>	Pointer to the array of features.
<i>pWeight</i>	Pointer to the array of feature weights.
<i>pThreshold</i>	Pointer to the array of classifier threshold values.
<i>pVal1, pVal2</i>	Pointers to the arrays of classifier result values.
<i>pNum</i>	Pointer to the array of classifier lengths.
<i>length</i>	Number of classifiers in the stage.

### Description

The function `ippiHaarClassifierInitAlloc` is declared in the `ippcv.h` file. This function allocates memory and initializes the structure for the sequence of Haar classifiers - classification stage. *i*-th classifier in the stage has *pNum*[*i*] rectangular features. Each feature is defined by the certain rectangle with horizontal and vertical sides. The length of vectors *pFeature*, *pWeight*, *pThreshold*, *pVal1*, and *pVal2* is equal to:

$$\sum_{i=0}^{length-1} pNum[i]$$

Result of applying classifiers to the image are computed in accordance with [formula](#) on page 14-55.

All features of the classifier initialized by the function `ippiHaarClassifierInitAlloc` have vertical and horizontal sides (left part of [Figure 14-8](#)). Some of these features later can be tilted by calling the function .

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>length</i> or one of <i>pNum[i]</i> is less than or equal to 0; or if the sum of all elements of <i>pNum</i> is not equal to <i>length</i> .
<code>ippStsBadArgErr</code>	Indicates an error condition if one of features is defined incorrectly.
<code>ippStsMemAllocErr</code>	Memory allocation error

---

## TiltedHaarClassifierInitAlloc

*Allocates memory and initializes the structure for tilted Haar classifiers.*

---

### Syntax

```
IppStatus ippiTiltedHaarClassifierInitAlloc_32f(IppiHaarClassifier_32f**
    pState, const IppiRect* pFeature, const Ipp32f* pWeight, const Ipp32f*
    pThreshold, const Ipp32f* pVal1, const Ipp32f* pVal2, const int* pNum, int
    length);

IppStatus ippiTiltedHaarClassifierInitAlloc_32s(IppiHaarClassifier_32s**
    pState, const IppiRect* pFeature, const Ipp32s* pWeight, const Ipp32s*
    pThreshold, const Ipp32s* pVal1, const Ipp32s* pVal2, const int* pNum, int
    length);
```

### Parameters

<i>ppState</i>	Pointer to the pointer to the Haar classifier structure.
<i>pFeature</i>	Pointer to the array of features.
<i>pWeight</i>	Pointer to the array of feature weights.

<i>pThreshold</i>	Pointer to the array of classifier threshold values.
<i>pVal1</i> , <i>pVal2</i>	Pointers to the arrays of classifier result values.
<i>pNum</i>	Pointer to the array of classifier lengths.
<i>length</i>	Number of classifiers in the stage.

### Description

The function `ippiHaarClassifierInitAlloc` is declared in the `ippcv.h` file. This function allocates memory and initializes the structure for the sequence of Haar classifiers - classification stage.  $t$ -th classifier in the stage has  $pNum[i]$  rectangular features. Each feature is defined by the certain rectangle with sides tilted by 45 degrees. The points with minimum and maximum row numbers should be specified. The length of vectors  $pFeature$ ,  $pWeight$ ,  $pThreshold$ ,  $pVal1$ , and  $pVal2$  is equal to:

$$\sum_{i=0}^{length-1} pNum[i]$$

Result of applying classifiers to the image are computed in accordance with [formula](#) on page 14-55.

All features of the classifier initialized by the function `ippiTiltedHaarClassifierInitAlloc` have tilted sides (right part of [Figure 14-8](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if $length$ or one of $pNum[i]$ is less than or equal to 0; or if the sum of all elements of $pNum$ is not equal to $length$ .
<code>ippStsBadArgErr</code>	Indicates an error condition if one of features is defined incorrectly.
<code>ippStsMemAllocErr</code>	Memory allocation error

---

## HaarClassifierFree

*Frees memory allocated for the Haar classifier structure.*

---

### Syntax

```
IppStatus ippiHaarClassifierFree_32f(IppiHaarClassifier_32f* pState);  
IppStatus ippiHaarClassifierFree_32s(IppiHaarClassifier_32s* pState);
```

### Parameters

*pState*                      Pointer to the Haar classifier structure.

### Description

The function `ippiHaarClassifierFree` is declared in the `ippcv.h` file. This function frees memory allocated for the Haar classifier structure *pState* by the function [ippiHaarClassifierInitAlloc](#) or [ippiTiltedHaarClassifierInitAlloc](#).

### Return Values

`ippStsNoErr`                Indicates no error.  
`ippStsNullPtrErr`        Indicates an error condition if the *pState* pointer is NULL.

---

## GetHaarClassifierSize

*Returns the size of the Haar classifier.*

---

### Syntax

```
IppStatus ippiGetHaarClassifierSize_32f(IppiHaarClassifier_32f* pState,  
    IppiSize* pSize);  
IppStatus ippiGetHaarClassifierSize_32s(IppiHaarClassifier_32s* pState,  
    IppiSize* pSize);
```

### Parameters

*pState*                      Pointer to the Haar classifier structure.

*pSize* Pointer to the size of Haar classifier structure.

### Description

The function `ippiGetHaarClassifierSize` is declared in the `ippcv.h` file. This function computes the minimum size of the window containing all features of the Haar classifier described by the *pState*.

### Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error condition if the *pState* pointer is NULL.

---

## TiltHaarFeatures

*Modifies a Haar classifier by tilting specified features.*

---

### Syntax

```
IppStatus ippiTiltHaarFeatures_32f(const Ipp8u* pMask, int flag,
    IppiHaarClassifier_32f* pState);
IppStatus ippiTiltHaarFeatures_32s(const Ipp8u* pMask, int flag,
    IppiHaarClassifier_32s* pState);
```

### Parameters

*pMask* Pointer to the mask vector.

*flag* Flag to choose the direction of feature tilting.

*pState* Pointer to the Haar classifier structure.

### Description

The function `ippiTiltHaarFeatures` is declared in the `ippcv.h` file. This function tilts specified features of the Haar classifier created by the function [ippiHaarClassifierInitAlloc](#). Non-zero elements of previously prepared vector *pMask* indicates the features that will be tilted. The *flag* parameter specifies how the features are tilted: if its value is equal to 0, the feature is tilted around the left top corner clockwise, if it is equal to 1, the feature is tilted around the bottom left corner counter-clockwise.

This mixed classifier containing both common and tilted features can be used by the function [`ippiApplyMixedHaarClassifier`](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointer is NULL.
<code>ippStsBadArgErr</code>	Indicates an error condition if the classifier is tilted already.

---

## ApplyHaarClassifier

*Applies a Haar classifier to an image.*

---

### Syntax

```
IppStatus ippiApplyHaarClassifier_32f_C1R(const Ipp32f* pSrc, int srcStep,
    const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize
    roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);

IppStatus ippiApplyHaarClassifier_32s32f_C1R(const Ipp32s* pSrc, int srcStep,
    const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize
    roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);

IppStatus ippiApplyHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc, int srcStep,
    const Ipp32s* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize
    roiSize, int* pPositive, Ipp32s threshold, IppiHaarClassifier_32s* pState,
    int scaleFactor);
```

### Parameters

<code>pSrc</code>	Pointer to the ROI in the source image of integrals.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pNorm</code>	Pointer to the ROI in the source image of norm factors.
<code>normStep</code>	Distance in bytes between starts of consecutive lines in the image of the norm factors.
<code>pMask</code>	Pointer to the source and destination image of classification decisions.
<code>maskStep</code>	Distance in bytes between starts of consecutive lines in the image of classification decisions.



<i>pPositive</i>	Pointer to the number of positive decisions.
<i>roiSize</i>	Size of the source and destination images ROI in pixels.
<i>threshold</i>	Stage threshold value.
<i>pState</i>	Pointer to the Haar classifier structure.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiApplyHaarClassifier` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the Haar classifier *pState* previously initialized by the function [ippiHaarClassifierInitAlloc](#) or [ippiTiltedHaarClassifierInitAlloc](#) to pixels of the source image ROI *pSrc*. The source image should be in the integral representation, it can be obtained by calling one of the [integral functions](#) beforehand. The sum of pixels on feature rectangles is computed as:

$$\sum_{l=1}^k (pSrc[i+y_l,j+x_l] - pSrc[i+Y_l,j+x_l] - pSrc[i+y_l,j+X_l] + pSrc[i+Y_l,j+X_l])w_l$$

Here  $(y_l, x_l)$  and  $(Y_l, X_l)$  are coordinates of top left and right bottom pixels of *l*-th rectangle of the feature, and  $w_l$  is the feature weight. For  $i=0...roiSize.height-1$ ,  $j=0...roiSize.width-1$  all pixels referred in the above formula should be allocated in memory.

The input value of `pPositive[0]` is used as a hint to choose the calculation algorithm. If it is greater than or equal to `roiSize.width*roiSize.height` the value of the classifier is calculated in accordance with the above formula for all pixels of the input image. Otherwise the value of the classifier is calculated for all non-zero pixels of *pMask* image. If the sum is less than *threshold* than the negative decision is made and the value of the corresponding pixel of the *pMask* image is set to zero. The number of positive decisions is assigned to the `pPositive[0]`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

<code>ippStsStepErr</code>	Indicates an error condition if one of the image step values is less than <code>roiSize.width*&lt;pixelSize&gt;</code> .
<code>ippStsNorEvenStepErr</code>	Indicates an error condition if one of the image step values is not divisible by 4 for 32-bit images.

---

## ApplyMixedHaarClassifier

*Applies a mixed Haar classifier to an image.*

---

### Syntax

```
IppStatus ippApplyMixedHaarClassifier_32f_C1R(const Ipp32f* pSrc, int srcStep,
const Ipp32f* pTilt, int tiltStep, const Ipp32f* pNorm, int normStep, Ipp8u*
pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold,
IppiHaarClassifier_32f* pState);

IppStatus ippApplyMixedHaarClassifier_32s32f_C1R(const Ipp32s* pSrc, int
srcStep, const Ipp32s* pTilt, int tiltStep, const Ipp32f* pNorm, int
normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive,
Ipp32f threshold, IppiHaarClassifier_32f* pState);

IppStatus ippApplyMixedHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc, int
srcStep, const Ipp32s* pTilt, int tiltStep, const Ipp32s* pNorm, int
normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive,
Ipp32s threshold, IppiHaarClassifier_32s* pState, int scaleFactor);
```

### Parameters

<code>pSrc</code>	Pointer to the ROI in the source image of integrals.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image of integrals.
<code>pTilt</code>	Pointer to the ROI in the source image of tilted integrals.
<code>tiltStep</code>	Distance in bytes between starts of consecutive lines in the source image of tilted integrals.
<code>pNorm</code>	Pointer to the ROI in the source image of norm factors.
<code>normStep</code>	Distance in bytes between starts of consecutive lines in the image of the norm factors.
<code>pMask</code>	Pointer to the source and destination image of classification decisions.

<i>maskStep</i>	Distance in bytes between starts of consecutive lines in the image of classification decisions.
<i>pPositive</i>	Pointer to the number of positive decisions.
<i>roiSize</i>	Size of the source and destination images ROI in pixels.
<i>threshold</i>	Stage threshold value.
<i>pState</i>	Pointer to the mixed Haar classifier structure.
<i>scaleFactor</i>	The factor for integer result scaling.

## Description

The function `ippiApplyMixedHaarClassifier` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function applies the mixed Haar classifier *pState* to the ROI of the source images *pSrc* and *pTilt*. The mixed Haar classifier is a classifier initialized by the function [ippiHaarClassifierInitAlloc](#) and then modified by the function [ippiTiltHaarFeatures](#). The source images must be in the integral representation, they can be obtained by calling one of the [integral functions](#) beforehand. Common features are applied to the *pSrc* image, and tilted features are applied to the *pTilt* image. The sum of pixels on feature rectangles is computed as:

$$\begin{aligned}
 & \sum_{l=1}^k (pSrc[i+y_lj+x_l] - pSrc[i+Y_lj+x_l] - pSrc[i+y_lj+X_l] + pSrc[i+Y_lj+X_l])w_l \\
 & \text{or} \\
 & \sum_{l=1}^k (pTilt[i+y_lj+x_l] - pTilt[i+Y_lj+x_l] - pTilt[i+y_lj+X_l] + pTilt[i+Y_lj+X_l])w_l
 \end{aligned}$$

Here  $(y_l, x_l)$  and  $(Y_l, X_l)$  are coordinates of top left and right bottom pixels of *l*-th rectangle of the feature, and  $w_l$  is the feature weight. For  $i = 0 \dots roiSize.height - 1$ ,  $j = 0 \dots roiSize.width - 1$  all pixels referred in the above formula should be allocated in memory.

The input value of `pPositive[0]` is used as a hint to choose the calculation algorithm. If it is greater than or equal to `roiSize.width*roiSize.height` the value of the classifier is calculated in accordance with the above formula for all pixels of the input image. Otherwise the

value of the classifier is calculated for all non-zero pixels of *pMask* image. If the sum is less than *threshold* than the negative decision is made and the value of the corresponding pixel of the *pMask* image is set to zero. The number of positive decisions is assigned to the *pPositive*[0].

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if one of the image step values is less than <i>roiSize.width*&lt;pixelSize&gt;</i> .
<code>ippStsNorEvenStepErr</code>	Indicates an error condition if one of the image step values is not divisible by 4 for 32-bit images.

## Camera Calibration and 3D Reconstruction

### Correction of Camera Lens Distortion

Digital camera usually introduces significant distortion caused by the camera and lens. These distortions cause errors in any analysis of the image. The functions described in this section correct these distortion using intrinsic camera parameters and distortion coefficients. These intrinsic camera parameters are focal lengths  $f_x$ ,  $f_y$ , and principal point coordinates  $c_x$ ,  $c_y$ . The distortion is characterized by two coefficients of radial distortions  $k_1$ ,  $k_2$  and two coefficients of tangential distortions  $p_1$ ,  $p_2$ .

The undistorted coordinates  $x_u$  and  $y_u$  of point with coordinates  $(x_d, y_d)$  are computed in accordance with the following formulas:

$$x_u = x_d \cdot (1 + k_1 r^2 + k_2 r^4) + 2p_1 x_d y_d + p_2 \cdot (r^2 + 2x_d^2)$$
$$y_u = y_d \cdot (1 + k_1 r^2 + k_2 r^4) + 2p_2 x_d y_d + p_1 \cdot (r^2 + 2y_d^2)$$

Here  $r^2 = x_d^2 + y_d^2$ ,  $x_d = (j - c_x)/f_x$ ,  $y_d = (i - c_y)/f_y$ ;  $i$  and  $j$  are row and columns numbers of the pixel. The pixel value is computed using bilinear interpolation of four nearest pixel of the source image. If undistorted coordinates are outside the image, then the destination pixel is not changed.

---

## UndistortGetSize

*Computes the size of the external buffer.*

---

### Syntax

```
IppStatus ippiUndistortGetSize(IppiSize roiSize, int* pBufSize);
```

### Parameters

<i>roiSize</i>	Size of source and destination images ROI in pixels.
<i>pBufSize</i>	Pointer to the computed value of the buffer size.

## Description

The function `ippiUndistortGetSize` is declared in the `ippcv.h` file. This function computes the size of the temporary external buffer that is used by the functions [ippiUndistortRadial](#). The buffer of the computed size can be used to process smaller images as well.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <code>pBufSize</code> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

---

## UndistortRadial

*Corrects radial distortions of the single image.*

---

## Syntax

```
IppStatus ippiUndistortRadial_<mod>(const Ipp<datatype>* pSrc, int  
    srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp32f  
    fx, Ipp32f fy, Ipp32f cx, Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp8u*  
    pBuffer);
```

Supported values for `mod` :

<code>8u_C1R</code>	<code>16u_C1R</code>	<code>32f_C1R</code>
<code>8u_C3R</code>	<code>16u_C3R</code>	<code>32f_C3R</code>

## Parameters

<code>pSrc</code>	Pointer to the ROI in the source distorted image.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the ROI in the destination corrected image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of source and destination images ROI in pixels.

<i>fx</i>	Focal lengths along the x axis.
<i>fy</i>	Focal lengths along the y axis.
<i>cx</i>	x-coordinate of the principal point.
<i>cy</i>	y-coordinate of the principal point.
<i>k1</i>	First coefficient of radial distortion.
<i>k2</i>	Second coefficient of radial distortion.
<i>pBuffer</i>	Pointer to the external buffer.

### Description

The function `ippiUndistortRadial` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function corrects radial distortions of the single source image *pSrc* and stores corrected image in the *pDst*. Correction is performed accounting camera parameters *fx*, *fy*, *cx*, *cy* and radial distortion parameters *k1*, *k2*. The function can also pass the pointer to the external buffer *pBuffer* whose size should be computed previously using the function [ippiUndistortGetSize](#).

If a null pointer is passed, slower computations without an external buffer will be performed.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pSrc</i> or <i>pDst</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> , or <i>dstStep</i> is less than <i>roiSize.width * &lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
<code>ippStsBadArgErr</code>	Indicates an error if <i>fx</i> or <i>fy</i> is equal to 0.

---

## CreateMapCameraUndistort

*Creates look-up tables of coordinates of corrected image.*

---

### Syntax

```
IppStatus ippiCreateMapCameraUndistort_32f_C1R(Ipp32f* pxMap, int xStep,
        Ipp32f* pyMap, int yStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy,
        Ipp32f cx, Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp32f p1, Ipp32f p2,
        Ipp8u* pBuffer);
```

### Parameters

<i>pxMap</i>	Pointer to the destination x coordinate look-up buffer.
<i>xStep</i>	Distance in bytes between starts of consecutive lines in the <i>pxMap</i> image.
<i>pyMap</i>	Pointer to the destination y coordinate look-up buffer.
<i>yStep</i>	Distance in bytes between starts of consecutive lines in the <i>pyMap</i> image.
<i>roiSize</i>	Size of source and destination images ROI in pixels.
<i>fx</i>	Focal lengths along the <i>x</i> axis.
<i>fy</i>	Focal lengths along the <i>y</i> axis.
<i>cx</i>	<i>x</i> - coordinate of the principal point.
<i>cy</i>	<i>y</i> - coordinate of the principal point.
<i>k1</i>	First coefficient of radial distortion.
<i>k2</i>	Second coefficient of radial distortion.
<i>p1</i>	First coefficient of tangential distortion.
<i>p2</i>	Second coefficient of tangential distortion.
<i>pBuffer</i>	Pointer to the external buffer.



## Description

The function `ippiCreateMapCameraUndistort` is declared in the `ippcv.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates the look-up tables of  $x$ - and  $y$ -coordinates `pxMap` and `pyMap` respectively. These coordinates are computed in accordance with camera parameters  $fx$ ,  $fy$ ,  $cx$ ,  $cy$ , and distortion parameters  $k1$ ,  $k2$ ,  $p1$ ,  $p2$ . The created tables can be used by the Intel IPP function [ippiRemap](#) to remap the distorted source image and get the corrected image.

To accelerate the computations the function can pass the pointer to the external buffer `pBuffer` whose size should be computed previously using the function [ippiUndistortGetSize](#). If a null pointer is passed, slower computations without an external buffer will be performed.

The following [Example 14-2](#) demonstrates how to correct camera lens distortion for set of images using Intel IPP functions.

---

### Example 14-2 Correction of Set of Distorted Images

---

```
Ipp32f *pxMap, *pyMap, *pBuffer;
Int xStep, yStep, buflen;
IppiSize roiSize;
IppiRect rect;

ippiUndistortGetSize (roiSize, &buflen);
buffer = ippiMalloc_8u(buflen);
xMap = ippiMalloc_32f(roiSize.x, roiSize.y, *xStep);
yMap = ippiMalloc_32f(roiSize.x, roiSize.y, *yStep);

ippiCreateMapCameraUndistort_32f_C1R (pxMap, pxStep, yMap, yStep, roiSize,
                                     fx, fy, cx, cy, k1, k2, 0, 0, pBuffer);
rect.x = 0;          rect.y = 0;
rect.width = roiSize.width; rect.height = roiSize.height;

...

ippiRemap_32f(pSrc, roiSize, srcStep, rect, pxMap, xStep, pyMap, yStep,
              pDst, dstStep, roiSize, IPPI_INTER_LINEAR);

...

ippiFree(yMap);
ippiFree(xMap);
ippsFree(buffer);
```

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pxMap</i> or <i>pyMap</i> is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if : <i>xStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> , or <i>yStep</i> is less than <i>roiSize.width</i> * <i>&lt;pixelSize&gt;</i> .
<code>ippStsNotEvenStepErr</code>	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
<code>ippStsBadArgErr</code>	Indicates an error when <i>fx</i> or <i>fy</i> is equal to 0.

# *Image Compression Functions*

---

15

This chapter describes Intel IPP image processing functions that prepare data and perform still image compression and coding in accordance with JPEG and JPEG2000 standards (see [\[ISO10918\]](#) and [\[ISO15444\]](#)).

These functions are grouped into the following sections:

[Support Functions](#)

## **Functions for JPEG coding:**

[Color Conversion Functions](#)

[Combined Color Conversion Functions](#)

[Quantization Functions](#)

[Combined DCT Functions](#)

[Level Shift Functions](#)

[Sampling Functions](#)

[Planar-to-Pixel and Pixel-to-Planar Conversion Functions](#)

[Huffman Codec Functions](#)

## **Lossless Mode of Operation**

[Functions for Lossless JPEG Coding](#)

## **Functions for JPEG2000 coding:**

[Wavelet Transform Functions](#)

[JPEG2000 Entropy Coding and Decoding Functions](#)

[Component Transform Functions](#)

Each section starts with a table that lists functions described in more detail later in this section, together with brief description of operations these functions perform.




---

**NOTE.** For simplicity, `ippi` prefix is omitted in function group names given in the summary tables. Function prototypes and references in the text contain full names.

---

The use of the Intel IPP JPEG functions is demonstrated in Intel IPP Samples. See *Intel IPP JPEG Processing Samples* downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>.

## Support Functions

This section describes the support function that returns information about the current version of the used Intel IPP JPEG codec software.

---

### ippjGetLibVersion

*Returns information of the library version.*

---

#### Syntax

```
const IppLibraryVersion* ippjGetLibVersion(void);
```

#### Description

The function `ippjGetLibVersion` is declared in the `ippj.h` file. This function returns a pointer to a static data structure `IppLibraryVersion` that contains information about the current version of the Intel IPP JPEG codec library. There is no need to free memory referenced by this pointer, as it points to a static variable. The `IppLibraryVersion` structure contains the following fields:

<i>major</i>	The major number of the current library version.
<i>minor</i>	The minor number of the current library version.
<i>Name</i>	The name of the library.
<i>Version</i>	The library version ASCII string.
<i>BuildDate</i>	The library version build date.

Return Values

The return value of the function is a pointer to a structure `IppLibraryVersion`.

Color Conversion Functions

This section describes Intel IPP color conversion functions that are specific for JPEG codec.

These functions are listed in the table [Table 15-1](#).

Table 15-1 Color Conversion Functions

Function Base Name	Description
<a href="#">RGBToY_JPEG</a>	Converts RGB images to gray scale.
<a href="#">BGRToY_JPEG</a>	Converts BGR images to gray scale.
<a href="#">RGBToYCbCr_JPEG</a>	Converts RGB images to the YCbCr color model.
<a href="#">YCbCrToRGB_JPEG</a>	Converts YCbCr images to the RGB color model.
<a href="#">RGB565ToYCbCr_JPEG,</a>	Convert 16-bit RGB images to the YCbCr color model.
<a href="#">RGB555ToYCbCr_JPEG</a>	
<a href="#">YCbCrToRGB565_JPEG,</a>	Convert YCbCr images to the 16-bit RGB images.
<a href="#">YCbCrToRGB555_JPEG</a>	
<a href="#">BGRToYCbCr_JPEG</a>	Converts BGR images to the YCbCr color model.
<a href="#">YCbCrToBGR_JPEG</a>	Converts YCbCr images to the BGR color model.
<a href="#">BGR565ToYCbCr_JPEG,</a>	Convert 16-bit BGR images to the YCbCr color model.
<a href="#">BGR555ToYCbCr_JPEG</a>	
<a href="#">YCbCrToBGR565_JPEG,</a>	Convert YCbCr images to the 16-bit BGR images.
<a href="#">YCbCrToBGR555_JPEG</a>	
<a href="#">CMYKToYCCK_JPEG</a>	Converts CMYK images to the YCCK color model.
<a href="#">YCCKToCMYK_JPEG</a>	Converts YCCK images to the CMYK color model.

---

## RGBToY\_JPEG

*Converts an RGB image to gray scale.*

---

### Syntax

#### Case 1: Operation on planar data

```
IppStatus ippRGBToY_JPEG_8u_P3C1R(const Ipp8u* pSrcRGB[3],  
    int srcStep, Ipp8u* pDstY, int dstStep, IppiSize roiSize);
```

#### Case 2: Operation on pixel-order data

```
IppStatus ippRGBToY_JPEG_8u_C3C1R(const Ipp8u* pSrcRGB, int srcStep,  
    Ipp8u* pDstY, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrcRGB</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstY</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippRGBToY_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts the RGB image to gray scale using the following formula:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.

<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.
----------------------------	--

---

## BGRTToY\_JPEG

*Converts a BGR image to gray scale.*

---

### Syntax

```
IppStatus ippBGRTToY_JPEG_8u_C3C1R(const Ipp8u* pSrcBGR, int srcStep,
                                     Ipp8u* pDstY, int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrcBGR</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstY</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippBGRTToY_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a BGR image to gray scale using the following formula:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.

`ippStsStepErr` Indicates an error condition if `srcStep` or `dstStep` has a zero or negative value.

---

## RGBToYCbCr\_JPEG

*Converts an RGB image to YCbCr color model.*

---

### Syntax

#### Case 1: Operation on planar data

```
IppStatus ippRGBToYCbCr_JPEG_8u_P3R(const Ipp8u* pSrcRGB[3], int
    srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

#### Case 2: Operation on pixel-order data

```
IppStatus ippRGBToYCbCr_JPEG_8u_C3P3R(const Ipp8u* pSrcRGB, int srcStep,
    Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrcRGB</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstYCbCr</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippRGBToYCbCr_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an RGB image to the YCbCr color model using the following formulas:

$$\begin{aligned} Y &= 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \\ Cb &= -0.16874 \cdot R - 0.33126 \cdot G + 0.5 \cdot B + 128 \\ Cr &= 0.5 \cdot R - 0.41869 \cdot G - 0.08131 \cdot B + 128 \end{aligned}$$



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

## YCbCrToRGB\_JPEG

*Converts an YCbCr image to the RGB color model.*

### Syntax

#### Case 1: Operation on planar data

```
IppStatus ippiYCbCrToRGB_JPEG_8u_P3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp8u* pDstRGB[3], int dstStep, IppiSize roiSize);
```

#### Case 2: Operation on pixel-order data

```
IppStatus ippiYCbCrToRGB_JPEG_8u_P3C3R(const Ipp8u* pSrcYCbCr[3], int
    srcStep, Ipp8u* pDstRGB, int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrcYCbCr</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstRGB</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippiYCbCrToRGB_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an YCbCr image to the RGB image according to the following formulas:

$$\begin{aligned} R &= Y + 1.402 * Cr - 179.456 \\ G &= Y - 0.34414 * Cb - 0.71414 * Cr + 135.45984 \\ B &= Y + 1.772 * Cb - 226.816 \end{aligned}$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

## RGB565ToYCbCr\_JPEG, RGB555ToYCbCr\_JPEG

*Convert an RGB image to YCbCr color model.*

---

### Syntax

```
ippStatus ippRGB565ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrcRGB,
    int srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
ippStatus ippRGB555ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrcRGB,
    int srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrcRGB</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstYCbCr</i>	An array of pointers to the ROIs in the separate destination color planes.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiRGB555ToYCbCr_JPEG` and `ippiRGB555ToYCbCr_JPEG` are declared in the `ippj.h` file. These functions convert an RGB image to the YCbCr image using the same formulas as in [ippiRGBToYCbCr\\_JPEG](#) function for computing Y, Cb, and Cr component values. The source image `pSrcRGB` has a reduced bit depth of 16 bits per pixel (see [Figure 6-14](#)), and it can be in one of two possible formats : RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), or RGB555 (5 bits for red, green, blue).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

---

## YCbCrToRGB565\_JPEG, YCbCrToRGB555\_JPEG

*Convert an YCbCr image to the RGB color model.*

---

## Syntax

```

IppStatus ippiYCbCrToRGB565_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3],
        int srcStep, Ipp16u* pDstRGB, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCrToRGB555_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3],
        int srcStep, Ipp16u* pDstRGB, int dstStep, IppiSize roiSize);

```

## Parameters

<code>pSrcYCbCr</code>	An array of pointers to the ROIs in the separate source color planes.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstRGB</code>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiYCbCrToRGB565_JPEG` and `ippiYCbCrToRGB565_JPEG` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert an YCbCr image to the RGB format using the same formulas as in [ippiYCbCrToRGB\\_JPEG](#) function for computing R, G, and B component values. The destination image *pDstRGB* has a reduced bit depth of 16 bits per pixel (see [Figure 6-14](#)), and it can be in one of two possible formats : RGB565 (5 bits for red, 6 bits for green, 5 bits for blue), or RGB555 (5 bits for red, green, blue).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

## BGRToYCbCr\_JPEG

*Converts a BGR image to YCbCr color model.*

---

### Syntax

```
ippStatus ippiBGRToYCbCr_JPEG_8u_C3P3R(const Ipp8u* pSrcBGR,  
    int srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrcBGR</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>pDstYCbCr</i>	An array of pointers to the ROIs in the separate destination color planes.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiBGRToYCbCr_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a BGR image to the YCbCr color model using the same formulas as in [ippiRGBToYCbCr\\_JPEG](#) function for computing Y, Cb, and Cr component values.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

## YCbCrToBGR\_JPEG

*Converts an YCbCr image to the BGR color model.*

---

### Syntax

```
ippStatus ippiYCbCrToBGR_JPEG_8u_P3C3R(const Ipp8u* pSrcYCbCr[3],
    int srcStep, Ipp8u* pDstBGR, int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrcYCbCr</i>	An array of pointers to the ROIs in the separate source color planes.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstBGR</i>	Pointer to the destination image ROI.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiYCbCrToBGR_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an YCbCr image to the BGR image using the same formulas as in [ippiYCbCrToRGB\\_JPEG](#) function for computing R, G, and B component values.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

## BGR565ToYCbCr\_JPEG, BGR555ToYCbCr\_JPEG

*Convert a BGR image to YCbCr color model.*

---

### Syntax

```
IppStatus ippiBGR565ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrcBGR, int
    srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
IppStatus ippiBGR555ToYCbCr_JPEG_16u8u_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp8u* pDstYCbCr[3], int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrcBGR</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.

<i>pDstYCbCr</i>	An array of pointers to the ROIs in the separate destination color planes.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The functions `ippiBGR555ToYCbCr_JPEG` and `ippiBGR555ToYCbCr_JPEG` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert an BGR image to the YCbCr image using the same formulas as in [ippiRGBToYCbCr\\_JPEG](#) function for computing Y, Cb, and Cr component values. The source image *pSrcBGR* has a reduced bit depth of 16 bits per pixel (see [Figure 6-14](#)), and it can be in one of two possible formats : BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

## YCbCrToBGR565\_JPEG, YCbCrToBGR555\_JPEG

*Convert an YCbCr image to the BGR color model.*

---

### Syntax

```
ippStatus ippiYCbCrToBGR565_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3],
        int srcStep, Ipp16u* pDstBGR, int dstStep, IppiSize roiSize);

ippStatus ippiYCbCrToBGR555_JPEG_8u16u_P3C3R(const Ipp8u* pSrcYCbCr[3],
        int srcStep, Ipp16u* pDstBGR, int dstStep, IppiSize roiSize);
```

## Parameters

<i>pSrcYCbCr</i>	An array of pointers to the ROIs in the separate source color planes.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstBGR</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The functions `ippiYCbCrToBGR565_JPEG` and `ippiYCbCrToBGR565_JPEG` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert an YCbCr image to the BGR format using the same formulas as in [ippiYCbCrToRGB\\_JPEG](#) function for computing R, G, and B component values. The destination image *pDstBGR* has a reduced bit depth of 16 bits per pixel (see [Figure 6-14](#)), and it can be in one of two possible formats : BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.



## CMYKToYCCK\_JPEG

*Converts a CMYK image to the YCCK color model.*

### Syntax

#### Case 1: Operation on planar data

```
IppStatus ippiCMYKToYCCK_JPEG_8u_P4R(const Ipp8u* pSrcCMYK[4],
    int srcStep, Ipp8u* pDstYCCK[4], int dstStep, IppiSize roiSize);
```

#### Case 2: Operation on pixel-order data

```
IppStatus ippiCMYKToYCCK_JPEG_8u_C4P4R(const Ipp8u* pSrcCMYK,
    int srcStep, Ipp8u* pDstYCCK[4], int dstStep, IppiSize roiSize);
```

### Parameters

<i>pSrcCMYK</i>	An array of pointers to the ROIs in the separate source color planes.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstYCCK</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiCMYKToYCCK_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a CMYK image to the YCCK image in two steps. First, conversion is done into RGB format:

$$\begin{aligned} R &= 255 - C \\ G &= 255 - M \\ B &= 255 - Y \end{aligned}$$

After that, conversion to YCCK image is performed as:

$$\begin{aligned} Y &= 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \\ Cb &= -0.16874 \cdot R - 0.33126 \cdot G + 0.5 \cdot B + 128 \\ Cr &= 0.5 \cdot R - 0.41869 \cdot G - 0.08131 \cdot B + 128 \end{aligned}$$

The values of  $\kappa$  channel are written without modification.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

---

## YCCKToCMYK\_JPEG

*Converts an YCCK image to the CMYK color model.*

---

### Syntax

```
IppStatus ippYCKToCMYK_JPEG_8u_P4R(const Ipp8u* pSrcYCCK[4],  
    int srcStep, Ipp8u* pDstCMYK[4], int dstStep, IppiSize roiSize);  
IppStatus ippYCKToCMYK_JPEG_8u_P4C4R(const Ipp8u* pSrcYCCK[4],  
    int srcStep, Ipp8u* pDstCMYK, int dstStep, IppiSize roiSize);
```

### Parameters

<code>pSrcYCCK</code>	An array of pointers to the ROIs in the separate source color planes.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstCMYK</code>	Pointer to the destination image ROI; an array of pointers to the ROIs in the separate destination color planes for planar image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

### Description

The function `ippYCKToCMYK_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an YCCK image to the CMYK image in two steps. First, conversion is done into RGB format as:

$$\begin{aligned}R &= Y + 1.402 * Cr - 179.456 \\G &= Y - 0.34414 * Cb - 0.71414 * Cr + 135.45984 \\B &= Y + 1.772 * Cb - 226.816\end{aligned}$$

After that, conversion to CMYK image is performed as follows:

$$\begin{aligned}C &= 255 - R \\M &= 255 - G \\Y &= 255 - B\end{aligned}$$

The values of  $\kappa$  channel are written without modification.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## Combined Color Conversion Functions

This section describes Intel IPP functions that perform sampling, color conversion and level shift (transition from unsigned data range to the signed one and vice versa) and are specific for JPEG codec. These functions are listed in the table [Table 15-2](#).

**Table 15-2 Combined Color Conversion Functions**

Function Base Name	Description
<a href="#">RGBToYCbCr444LS MCU</a>	Converts RGB images to YCbCr color model and creates 444 MCU.
<a href="#">RGBToYCbCr422LS MCU</a>	Converts RGB images to YCbCr color model and creates 422 MCU.
<a href="#">RGBToYCbCr411LS MCU</a>	Converts RGB images to YCbCr color model and creates 411 MCU.
<a href="#">BGRToYCbCr444LS MCU</a>	Converts BGR images to YCbCr color model and creates 444 MCU.
<a href="#">BGR565ToYCbCr444LS MCU,</a>	Convert 16-bit BGR images to YCbCr color model and create 444 MCU.
<a href="#">BGR555ToYCbCr444LS MCU</a>	
<a href="#">BGRToYCbCr422LS MCU</a>	Converts BGR images to YCbCr color model and creates 422 MCU.
<a href="#">BGR565ToYCbCr422LS MCU,</a>	Convert 16-bit BGR images to YCbCr color model and create 422 MCU
<a href="#">BGR555ToYCbCr422LS MCU</a>	
<a href="#">BGRToYCbCr411LS MCU</a>	Converts BGR images to YCbCr color model and creates 411 MCU.
<a href="#">BGR565ToYCbCr411LS MCU,</a>	Convert 16-bit BGR images to YCbCr color model and create 411 MCU
<a href="#">BGR555ToYCbCr411LS MCU</a>	
<a href="#">CMYKToYCK444LS MCU</a>	Converts CMYK images to YCK color model and creates 4444 MCU.
<a href="#">CMYKToYCK422LS MCU</a>	Converts CMYK images to YCK color model and creates 4224 MCU.
<a href="#">CMYKToYCK411LS MCU</a>	Converts CMYK images to YCK color model and creates 4114 MCU.
<a href="#">YCbCr444ToRGBLS MCU</a>	Creates YCbCr image from 444 MCU and converts it to RGB color model.
<a href="#">YCbCr422ToRGBLS MCU</a>	Creates YCbCr image from 422 MCU and converts it to RGB color model.
<a href="#">YCbCr411ToRGBLS MCU</a>	Creates YCbCr image from 411 MCU and converts it to RGB color model.
<a href="#">YCbCr444ToBGRLS MCU</a>	Creates YCbCr image from 444 MCU and converts it to BGR color model.
<a href="#">YCbCr444ToBGR565LS MCU,</a>	Create YCbCr image from 444 MCU and convert it to 16-bit BGR image.
<a href="#">YCbCr444ToBGR555LS MCU</a>	
<a href="#">YCbCr422ToBGRLS MCU</a>	Creates YCbCr image from 422 MCU and converts it to BGR color model.
<a href="#">YCbCr422ToBGR565LS MCU,</a>	Create YCbCr image from 422 MCU and convert it to 16-bit BGR image.
<a href="#">YCbCr422ToBGR555LS MCU</a>	

Table 15-2 Combined Color Conversion Functions (continued)

Function Base Name	Description
<a href="#">YCbCr411ToBGRLS_MCU</a>	Creates YCbCr image from 411 MCU and converts it to BGR color model.
<a href="#">YCbCr411ToBGR565LS_MCU,</a> <a href="#">YCbCr411ToBGR555LS_MCU</a>	Create YCbCr image from 411 MCU and convert it to 16-bit BGR image.
<a href="#">YCK444ToCMYKLS_MCU</a>	Creates YCK image from 4444 MCU and converts it to CMYK color model.
<a href="#">YCK422ToCMYKLS_MCU</a>	Creates YCK image from 4224 MCU and converts it to CMYK color model.
<a href="#">YCK411ToCMYKLS_MCU</a>	Creates YCK image from 4114 MCU and converts it to CMYK color model.

Here MCU stands for JPEG Minimum Coded Unit, that is, an interleaved set of blocks of size determined by the sampling factors (see [Figure 6-13](#)), or a single block in a non interleaved scan.

All functions implement the level shift operation in accordance with its definition by [[ISO10918](#)], Annex A.3.1, Level Shift.

## RGBToYCbCr444LS\_MCU

Converts an RGB image to the YCbCr color model and creates 444 MCU.

### Syntax

```
IppStatus ippiRGBToYCbCr444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB,
        int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<i>pSrcRGB</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Array of 3 pointers to the destination image blocks.

## Description

The function `ippiRGBToYCbCr444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an RGB image to the YCbCr image using the same formulas as in [ippiRGBToYCbCr\\_JPEG](#) function for computing Y, Cb, and Cr component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 444 MCU (see [Figure 6-13](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

---

## RGBToYCbCr422LS\_MCU

*Converts an RGB image to the YCbCr color model and creates 422 MCU.*

---

## Syntax

```
IppStatus ippiRGBToYCbCr422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB,  
int srcStep, Ipp16s* pDstMCU[3]);
```

## Parameters

<code>pSrcRGB</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 3 pointers to the destination image blocks.

## Description

The function `ippiRGBToYCbCr422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an RGB image to the YCbCr image using the same formulas as in [ippiRGBToYCbCr\\_JPEG](#) function for computing Y, Cb, and Cr component values.

Additionally, this function converts source data from unsigned Ipp8u range [0..255] to the signed Ipp16s range [-128..127] performing level shift operation (by subtracting 128) and creates 422 MCU (see [Figure 6-13](#)).

Downsampling is performed by plain averaging.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

---

## RGBToYCbCr411LS\_MCU

*Converts an RGB image to the YCbCr color model and creates 411 MCU.*

---

### Syntax

```
IppStatus ippiRGBToYCbCr411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcRGB,  
int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<code>pSrcRGB</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 3 pointers to the destination image blocks.

### Description

The function `ippiRGBToYCbCr411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function converts an RGB image to the YCbCr image using the same formulas as in [ippiRGBToYCbCr\\_JPEG](#) function for computing Y, Cb, and Cr component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 411 MCU (see [Figure 6-13](#)).

Downsampling is performed by plain averaging.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

---

## BGRToYCbCr444LS\_MCU

*Converts a BGR image to the YCbCr color model and creates 444 MCU.*

---

### Syntax

```
IppStatus ippIBGRToYCbCr444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR,  
int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<code>pSrcBGR</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 3 pointers to the destination image blocks.

### Description

The function `ippIBGRToYCbCr444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function converts a BGR image to the YCbCr image using the same formulas as in [ippiRGBToYCbCr\\_JPEG](#) function for computing `Y`, `Cb`, and `Cr` component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 444 MCU (see [Figure 6-13](#)).



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

## BGR565ToYCbCr444LS\_MCU, BGR555ToYCbCr444LS\_MCU

*Convert a BGR image to the YCbCr color model and create 444 MCU.*

### Syntax

```
ippStatus ippBGR565ToYCbCr444LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);

ippStatus ippBGR555ToYCbCr444LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<code>pSrcBGR</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 3 pointers to the destination image blocks.

### Description

The functions `ippBGR565ToYCbCr444LS_MCU` and `ippBGR555ToYCbCr444LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a source BGR image to the YCbCr 444 MCU (see [Figure 6-13](#)). The source image `pSrcBGR` has a reduced bit depth of 16 bits per pixel (see [Figure 6-14](#)), and it can be in one of two possible formats: BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

Source data are converted to the signed `Ipp16s` range `[-128..127]` by transforming them at first to the unsigned `Ipp8u` range with following level shift operation (by subtracting 128). `Y`, `Cb`, and `Cr` component values are computed using the same formulas as in [ippiRGBToYCbCr\\_JPEG](#) function.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

---

## BGRToYCbCr422LS\_MCU

*Converts an BGR image to the YCbCr color model and creates 422 MCU.*

---

### Syntax

```
IppStatus ippiBGRToYCbCr422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR,
        int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<code>pSrcBGR</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 3 pointers to the destination image blocks.

### Description

The function `ippiBGRToYCbCr422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an BGR image to the YCbCr image using the same formulas as in [ippiRGBToYCbCr\\_JPEG](#) function for computing `Y`, `Cb`, and `Cr` component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 422 MCU (see [Figure 6-13](#)).

Downsampling is performed by plain averaging.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

---

## BGR565ToYCbCr422LS\_MCU, BGR555ToYCbCr422LS\_MCU

*Convert an BGR image to the YCbCr color model and create 422 MCU.*

---

### Syntax

```
IppStatus ippIBGR565ToYCbCr422LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
IppStatus ippIBGR555ToYCbCr422LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<code>pSrcBGR</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 3 pointers to the destination image blocks.

### Description

The functions `ippIBGR565ToYCbCr422LS_MCU` and `ippIBGR555ToYCbCr422LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a source BGR image to the YCbCr 422 MCU (see [Figure 6-13](#)). The source image `pSrcBGR` has a reduced bit depth of 16 bits per pixel (see [Figure 6-14](#)), and it can be in one of two possible formats : BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

Source data are converted to the signed `Ipp16s` range `[-128..127]` by transforming them at first to the unsigned `Ipp8u` range with following level shift operation (by subtracting 128). `Y`, `Cb`, and `Cr` component values are computed using the same formulas as in [ippiRGBToYCbCr\\_JPEG](#) function.

Downsampling is performed by plain averaging.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

---

## BGRToYCbCr411LS\_MCU

*Converts an BGR image to the YCbCr color model and creates 411 MCU.*

---

### Syntax

```
IppStatus ippiBGRToYCbCr411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrcBGR,  
int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<code>pSrcBGR</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 3 pointers to the destination image blocks.

### Description

The function `ippiBGRToYCbCr411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts an BGR image to the YCbCr image using the same formulas as in [ippiRGBToYCbCr\\_JPEG](#) function for computing `Y`, `Cb`, and `Cr` component values.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 411 MCU (see [Figure 6-13](#)).

Downsampling is performed by plain averaging.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

---

## BGR565ToYCbCr411LS\_MCU, BGR555ToYCbCr411LS\_MCU

*Convert an BGR image to the YCbCr color model and create 411 MCU.*

---

### Syntax

```
IppStatus ippkBGR565ToYCbCr411LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
IppStatus ippkBGR555ToYCbCr411LS_MCU_16u16s_C3P3R(const Ipp16u* pSrcBGR,
    int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<code>pSrcBGR</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 3 pointers to the destination image blocks.

### Description

The functions `ippkBGR565ToYCbCr411LS_MCU` and `ippkBGR555ToYCbCr411LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)).

These functions convert a source BGR image to the YCbCr 411 MCU (see [Figure 6-13](#)). The source image *pSrcBGR* has a reduced bit depth of 16 bits per pixel (see [Figure 6-14](#)), and it can be in one of two possible formats : BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

Source data are converted to the signed *Ipp16s* range [-128..127] by transforming them at first to the unsigned *Ipp8u* range with following level shift operation (by subtracting 128). Y, Cb, and Cr component values are computed using the same formulas as in [ippiRGBToYCbCr JPEG](#) function.

Downsampling is performed by plain averaging.

### Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if any of the specified pointers is NULL.
<i>ippStsStepErr</i>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

---

## CMYKToYCK444LS\_MCU

*Converts a CMYK image to the YCK color model and creates 4444 MCU.*

---

### Syntax

```
IppStatus ippiCMYKToYCK444LS_MCU_8u16s_C4P4R(const Ipp8u* pSrcCMYK,  
int srcStep, Ipp16s* pDstMCU[4]);
```

### Parameters

<i>pSrcCMYK</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Array of 4 pointers to the destination image blocks.

### Description

The function `ippiCMYKToYCK444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a CMYK image to the YCCK image in the same way as the function [ippiCMYKToYCCK\\_JPEG](#) does.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 4444 MCU.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

---

## CMYKToYCCK422LS\_MCU

*Converts a CMYK image to the YCCK color model and creates 4224 MCU.*

---

### Syntax

```
IppStatus ippiCMYKToYCCK422LS_MCU_8u16s_C4P4R(const Ipp8u* pSrcCMYK,  
int srcStep, Ipp16s* pDstMCU[4]);
```

### Parameters

<code>pSrcCMYK</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 4 pointers to the destination image blocks.

### Description

The function `ippiCMYKToYCCK422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts a CMYK image to the YCCK image in the same way as the function [ippiCMYKToYCCK\\_JPEG](#) does.

Additionally, this function converts source data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 4224 MCU.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

---

## CMYKToYCCK411LS\_MCU

*Converts a CMYK image to the YCCK color model and creates 4114 MCU.*

---

### Syntax

```
IppStatus ippCMYKToYCCK411LS_MCU_8u16s_C4P4R(const Ipp8u* pSrcCMYK,  
        int srcStep, Ipp16s* pDstMCU[4]);
```

### Parameters

<code>pSrcCMYK</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstMCU</code>	Array of 4 pointers to the destination image blocks.

### Description

The function `ippCMYKToYCCK411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function converts a CMYK image to the YCCK image in the same way as the function [ippCMYKToYCCK JPEG](#) does.

Additionally, this function converts data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128) and creates 4114 MCU.

Downsampling is performed by plain averaging.



### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

## YCbCr444ToRGBLS\_MCU

*Creates an YCbCr image from 444 MCU and converts it to the RGB color model.*

### Syntax

```
IppStatus ippYCbCr444ToRGBLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
        Ipp8u* pDstRGB, int dstStep);
```

### Parameters

<code>pSrcMCU</code>	Array of 3 pointers to the source image blocks.
<code>pDstRGB</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiYCbCr444ToRGBLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCbCr 8x8 image from 444 MCU and converts it to the RGB format using the same formulas as in [ippiYCbCrToRGB\\_JPEG](#) function for computing R, G, and B component values.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> has a zero or negative value.

---

## YCbCr422ToRGBLS\_MCU

*Creates an YCbCr image from 422 MCU and converts it to the RGB color model.*

---

### Syntax

```
IppStatus ippiYCbCr422ToRGBLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
        Ipp8u* pDstRGB, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstRGB</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiYCbCr422ToRGBLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCbCr 16x8 image from 422 MCU and converts it to the RGB format using the same formulas as in [ippiYCbCrToRGB\\_JPEG](#) function for computing R, G, and B component values.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Upsampling is performed by replication of the neighbor value.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

---

## YCbCr411ToRGBLS\_MCU

*Creates an YCbCr image from 411 MCU and converts it to the RGB color model.*

---

### Syntax

```
IppStatus ippiYCbCr411ToRGBLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],
        Ipp8u* pDstRGB, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstRGB</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiYCbCr411ToRGBLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCbCr 16x16 image from 411 MCU and converts it to the RGB format using the same formulas as in [ippiYCbCrToRGB\\_JPEG](#) function for computing R, G, and B component values.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

Upsampling is performed by replication of the neighbor value.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

---

## YCbCr444ToBGRLS\_MCU

*Creates an YCbCr image from 444 MCU and converts it to the BGR color model.*

---

### Syntax

```
IppStatus ippiYCbCr444ToBGRLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
        Ipp8u* pDstBGR, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstBGR</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiYCbCr444ToBGRLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCbCr 8x8 image from 444 MCU and converts it to the BGR format using the same formulas as in [ippiYCbCrToRGB\\_JPEG](#) function for computing B, G, and R component values.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

## YCbCr444ToBGR565LS\_MCU, YCbCr444ToBGR555LS\_MCU

Create an YCbCr image from 444 MCU and convert it to the BGR color model.

### Syntax

```
IppStatusippiYCbCr444ToBGR565LS_MCU_16s16u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp16u* pDstBGR, int dstStep);

IppStatusippiYCbCr444ToBGR555LS_MCU_16s16u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp16u* pDstBGR, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstBGR</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The functions `ippiYCbCr444ToBGR565LS_MCU` and `ippiYCbCr444ToBGR555LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)). These functions create YCbCr 8x8 image from 444 MCU (see [Figure 6-13](#)) and convert it to the BGR format using the same formulas as in [ippiYCbCrToRGB\\_JPEG](#) function for computing B, G, and R component values.

Additionally, the functions convert data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128). The destination image *pDstBGR* has a reduced bit depth of 16 bits per pixel (see [Figure 6-14](#)), and it can be in one of two possible formats: BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

---

## YCbCr422ToBGRLS\_MCU

*Creates a YCbCr image from 422 MCU and converts it to the BGR color model.*

---

### Syntax

```
IppStatus ippiYCbCr422ToBGRLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
        Ipp8u* pDstBGR, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstBGR</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiYCbCr422ToBGRLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCbCr 16x8 image from 422 MCU (see [Figure 6-13](#)) and converts it to the BGR format using the same formulas as in [ippiYCbCrToRGB\\_JPEG](#) function for computing B, G, and R component values. Upsampling is performed by replication of the neighbor value.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

## YCbCr422ToBGR565LS\_MCU, YCbCr422ToBGR555LS\_MCU

Create an YCbCr image from 422 MCU and convert it to the BGR color model.

### Syntax

```
IppStatusippiYCbCr422ToBGR565LS_MCU_16s16u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp16u* pDstBGR, int dstStep);

IppStatusippiYCbCr422ToBGR555LS_MCU_16s16u_P3C3R(const Ipp16s* pSrcMCU[3],
    Ipp16u* pDstBGR, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstBGR</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The functions `ippiYCbCr444ToBGR565LS_MCU` and `ippiYCbCr444ToBGR555LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)). These functions create YCbCr 8x8 image from 422 MCU (see [Figure 6-13](#)) and convert it to the BGR format using the same formulas as in [ippiYCbCrToRGB\\_JPEG](#) function for computing B, G, and R component values.

Upsampling is performed by replication of the neighbor value.

Additionally, the functions convert data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

The destination image *pDstBGR* has a reduced bit depth of 16 bits per pixel (see [Figure 6-14](#)), and it can be in one of two possible formats: BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.

`ippStsStepErr` Indicates an error condition if *dstStep* has a zero or negative value.

---

## YCbCr411ToBGRLS\_MCU

*Creates an YCbCr image from 411 MCU and converts it to the BGR color model.*

---

### Syntax

```
IppStatus ippYCbCr411ToBGRLS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
        Ipp8u* pDstBGR, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstBGR</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiYCbCr411ToBGRLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCbCr 16x16 image from 411 MCU and converts it to the BGR format using the same formulas as in [ippiYCbCrToRGB\\_JPEG](#) function for computing B, G, and R component values. Upsampling is performed by replication of the neighbor value.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.



## YCbCr411ToBGR565LS\_MCU, YCbCr411ToBGR555LS\_MCU

Create an YCbCr image from 411 MCU and convert it to the BGR color model.

### Syntax

```
IppStatusippiYCbCr411ToBGR565LS_MCU_16s16u_P3C3R(constIpp16s*pSrcMCU[3],
    Ipp16u* pDstBGR, int dstStep);
IppStatusippiYCbCr411ToBGR555LS_MCU_16s16u_P3C3R(constIpp16s*pSrcMCU[3],
    Ipp16u* pDstBGR, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Array of 3 pointers to the source image blocks.
<i>pDstBGR</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The functions `ippiYCbCr411ToBGR565LS_MCU` and `ippiYCbCr411ToBGR555LS_MCU` are declared in the `ippj.h` file. They operate with ROI (see [Regions of Interest in Intel IPP](#)). These functions create YCbCr 8x8 image from 411 MCU (see [Figure 6-13](#)) and convert it to the BGR format using the same formulas as in [ippiYCbCrToRGB\\_JPEG](#) function for computing B, G, and R component values.

Upsampling is performed by replication of the neighbor value.

Additionally, the functions convert data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

The destination image *pDstBGR* has a reduced bit depth of 16 bits per pixel (see [Figure 6-14](#)), and it can be in one of two possible formats: BGR565 (5 bits for blue, 6 bits for green, 5 bits for red), or BGR555 (5 bits for blue, green, red).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.

`ippStsStepErr` Indicates an error condition if `dstStep` has a zero or negative value.

---

## YCKK444ToCMYKLS\_MCU

*Creates YCKK image from 4444 MCU and converts it to the CMYK color model.*

---

### Syntax

```
IppStatus ippYCKK444ToCMYKLS_MCU_16s8u_P4C4R(const Ipp16s* pSrcMCU[4],  
        Ipp8u* pDstCMYK, int dstStep);
```

### Parameters

<code>pSrcMCU</code>	Array of 4 pointers to the source image blocks.
<code>pDstCVBYK</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippYCKK444ToCMYKLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCKK 8x8 image from 4444 MCU and converts it to the CMYK color model in the same way as the function [ippYCKKToCMYK\\_JPEG](#) does.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> has a zero or negative value.

---

## YCCK422ToCMYKLS\_MCU

*Creates an YCCK image from 4224 MCU and converts it to the CMYK color model.*

---

### Syntax

```
IppStatus ippiYCCK422ToCMYKLS_MCU_16s8u_P4C4R(const Ipp16s* pSrcMCU[4],  
        Ipp8u* pDstCMYK, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Array of 4 pointers to the source image blocks.
<i>pDstCVBYK</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiYCCK422ToCMYKLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCCK 16x8 image from 4224 MCU and converts it to the CMYK color model in the same way as the function [ippiYCCKToCMYK\\_JPEG](#) does. Upsampling is performed by replication of the neighbor value.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

---

## YCK411ToCMYKLS\_MCU

*Creates an YCK image from 4114 MCU and converts it to the CMYK color model.*

---

### Syntax

```
IppStatus ippiYCK411ToCMYKLS_MCU_16s8u_P4C4R(const Ipp16s* pSrcMCU[4],  
        Ipp8u* pDstCMYK, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Array of 4 pointers to the source image blocks.
<i>pDstCVBYK</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiYCK411ToCMYKLS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)). This function creates YCK 16x16 image from 4114 MCU and converts it to the CMYK color model in the same way as the function [ippiYCKToCMYK\\_JPEG](#) does. Upsampling is performed by replication of the neighbor value.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

## Quantization Functions

This section describes Intel IPP functions that perform quantization, dequantization and preparation of quantization tables and are specific for JPEG codec. These functions are listed in the [Table 15-3](#).

**Table 15-3      Quantization Functions**

Function Base Name	Description
<a href="#">QuantFwdRawTableInit_JPEG</a>	Modifies raw quantization tables in accordance with quality factor.
<a href="#">QuantFwdTableInit_JPEG</a>	Prepares quantization tables in a format suitable for an encoder.
<a href="#">QuantFwd8x8_JPEG</a>	Performs quantization of an 8x8 block of DCT coefficients.
<a href="#">QuantInvTableInit_JPEG</a>	Prepares quantization tables in a format suitable for a decoder.
<a href="#">QuantInv8x8_JPEG</a>	Performs dequantization of DCT coefficients in 8x8 block

### QuantFwdRawTableInit\_JPEG

*Modifies raw quantization tables in accordance with the quality factor.*

#### Syntax

```
IppStatus ippiQuantFwdRawTableInit_JPEG_8u(Ipp8u* pQuantRawTable,  
      int qualityFactor);
```

#### Parameters

*pQuantRawTable*      Pointer to the raw quantization table.

*qualityFactor*        JPEG quality factor, must be in the range [1..100].

### Description

The function `ippiQuantFwdRawTableInit_JPEG` is declared in the `ippj.h` file. This function modifies the initial raw quantization table in accordance with the quality factor using the algorithm which is specified in [IJG JPEG Library, version 6b](#). This function is optional for JPEG codec implementation.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers are NULL.

---

## QuantFwdTableInit\_JPEG

*Prepares quantization tables suitable for fast encoding.*

---

### Syntax

```
IppStatus ippiQuantFwdTableInit_JPEG_8u16u(  
    const Ipp8u* pQuantRawTable, Ipp16u* pQuantFwdTable);
```

### Parameters

<code>pQuantRawTable</code>	Pointer to the raw quantization table.
<code>pQuantFwdTable</code>	Pointer to the quantization table for the encoder.

### Description

The function `ippiQuantFwdTableInit_JPEG` is declared in the `ippj.h` file. This function prepares quantization table in a format that is suitable for fast encoding. Initial raw quantization tables have zigzag order by definition. This function performs reordering transformation that converts the zigzag sequence of table elements to conventional order (left-to-right, top-to-bottom). To avoid division during quantization, the function `ippiQuantFwdTableInit_JPEG` scales the array by 15 bits.

The pointer `pQuantRawTable` points to the array of 64 elements as is required by [\[ISO10918\]](#), Annex B.2.4.1, *Quantization Table Specification Syntax*.

The pointer `pQuantFwdTable` points to an array containing 64 values of `Ipp16u` type.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.

## QuantFwd8x8\_JPEG

*Performs quantization of an 8x8 block of DCT coefficients.*

### Syntax

```
IppStatus ippQuantFwd8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*
    pQuantFwdTable);
```

### Parameters

<code>pSrcDst</code>	Pointer to the 8x8 block of DCT coefficients.
<code>pQuantFwdTable</code>	Pointer to the quantization table for the encoder.

### Description

The function `ippQuantFwd8x8_JPEG` is declared in the `ippj.h` file. This function performs quantization of computed DCT coefficients for an 8x8 block. Quantization is defined in [\[ISO10918\]](#), Annex A.3.4, *DCT Coefficient Quantization and Dequantization*, as

$$Sq(u, v) = \text{round}(S(u, v) / Q(u, v)),$$

where  $Sq(u, v)$  is the quantized DCT coefficient,  
 $S(u, v)$  is a computed DCT coefficient, and  
 $Q(u, v)$  is a quantization table element.

Rounding is to the nearest integer. To avoid division, quantization table elements are scaled up by  $2^{15}$ . After that, division operation is replaced by multiplication of DCT coefficients by scaled quantization table elements. Then the products are rescaled back by 15 bits.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both specified pointers are NULL.

---

## QuantInvTableInit\_JPEG

*Prepares quantization tables suitable for fast decoding.*

---

### Syntax

```
IppStatus ippiQuantInvTableInit_JPEG_8u16u(const Ipp8u* pQuantRawTable,  
      Ipp16u* pQuantInvTable);
```

### Parameters

*pQuantRawTable*      Pointer to the raw quantization table.

*pQuantInvTable*      Pointer to the quantization table for decoding.

### Description

The function `ippiQuantInvTableInit_JPEG` is declared in the `ippj.h` file. This function prepares a quantization table for fast decoding. It also performs reordering transformation that converts the zigzag sequence of table elements to conventional order (left-to-right, top-to-bottom) that is more suitable for a decoder. See [[ISO10918](#)], *Annex A.3.6, Zig-zag Sequence* for more information on zigzag order.

### Return Values

`ippStsNoErr`            Indicates no error.

`ippStsNullPtrErr`      Indicates an error condition if one or both of the specified pointers are NULL.

---

## QuantInv8x8\_JPEG

*Performs dequantization of an 8x8 block of DCT coefficients*

---

### Syntax

```
IppStatus ippiQuantInv8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*  
      pQuantInvTable);
```



### Parameters

<i>pSrcDst</i>	Pointer to the DCT coefficients of the 8x8 block.
<i>pQuantInvTable</i>	Pointer to the quantization table for decoding.

### Description

The function `ippiQuantInv8x8_JPEG` is declared in the `ippj.h` file. This function performs dequantization of the 8x8 block of DCT coefficients. Dequantization operation is defined in [\[ISO10918\]](#), *Annex A.3.4, DCT Coefficient Quantization and Dequantization*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.

## Combined DCT Functions

This section describes functions that perform Discrete Cosine Transform (DCT) plus some additional operations - level shift, quantization (or dequantization) - and are specific for JPEG codec. These functions are listed in [Table 15-4](#).

**Table 15-4**      **Functions that combine Quantization, DCT, and Level Shift.**

Function Base Name	Description
<a href="#">DCTQuantFwd8x8_JPEG</a>	Performs forward DCT and quantization.
<a href="#">DCTQuantFwd8x8_JPEG</a>	Performs forward DCT, quantization and level shift.
<a href="#">DCTQuantInv8x8_JPEG</a>	Performs inverse DCT and dequantization.
<a href="#">DCTQuantInv8x8LS_JPEG</a>	Performs inverse DCT, dequantization and level shift.

These functions implement component operations in accordance with their definition by [\[ISO10918\]](#), specifically:

- for DCT, as in [\[ISO10918\]](#), *Annex A.3.3, FDCT and IDCT*.
- for quantization/dequantization, as in [\[ISO10918\]](#), *Annex A.3.4, DCT Coefficient Quantization and Dequantization*;
- for level shift, as in [\[ISO10918\]](#), *Annex A.3.1, Level Shift*;

## DCTQuantFwd8x8\_JPEG

*Performs forward DCT and quantization of an 8x8 block.*

### Syntax

```

IppStatus ippiDCTQuantFwd8x8_JPEG_16s_C1(const Ipp16s* pSrc, Ipp16s*
    pDst, const Ipp16u* pQuantFwdTable);

IppStatus ippiDCTQuantFwd8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*
    pQuantFwdTable);

```

### Parameters

*pSrc*                                      Pointer to the 8x8 block in the source image.

<i>pDst</i>	Pointer to the destination 8x8 block of the quantized DCT coefficients.
<i>pSrcDst</i>	Pointer to the source and destination 8x8 block for in-place operation.
<i>pQuantFwdTable</i>	Pointer to the quantization table for the encoder.

### Description

The function `ippiDCTQuantFwd8x8_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a forward DCT and quantization of DCT coefficients.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .

---

## DCTQuantFwd8x8LS\_JPEG

*Performs level shift, forward DCT and quantization of an 8x8 block.*

---

### Syntax

```
ippiStatus ippiDCTQuantFwd8x8LS_JPEG_8u16s_C1R(const Ipp8u* pSrc,
        int srcStep, Ipp16s* pDst, const Ipp16u* pQuantFwdTable);
```

### Parameters

<i>pSrc</i>	Pointer to the 8x8 block in the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination 8x8 block of the quantized DCT coefficients.
<i>pQuantFwdTable</i>	Pointer to the quantization table for the encoder.

### Description

The function `ippiDCTQuantFwd8x8LS_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a forward DCT, quantization of DCT coefficients, and data conversion from the unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .

---

## DCTQuantInv8x8\_JPEG

*Performs dequantization and inverse DCT of an 8x8 block.*

---

### Syntax

```
IppStatus ippIDCTQuantInv8x8_JPEG_16s_C1(const Ipp16s* pSrc, Ipp16s*  
    pDst, const Ipp16u* pQuantInvTable);  
IppStatus ippIDCTQuantInv8x8_JPEG_16s_C1I(Ipp16s* pSrcDst, const Ipp16u*  
    pQuantInvTable);
```

### Parameters

<code>pSrc</code>	Pointer to the 8x8 block of quantized DCT coefficients.
<code>pDst</code>	Pointer to the 8x8 block in the destination image.
<code>pSrcDst</code>	Pointer to the source and destination 8x8 block for in-place operation.
<code>pQuantInvTable</code>	Pointer to the quantization table for the decoder.

### Description

The function `ippIDCTQuantInv8x8_JPEG` is declared in the `ippj.h` file. This function performs an inverse DCT and dequantization of DCT coefficients.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition when any of the specified pointers is <code>NULL</code> .

---

## DCTQuantInv8x8LS\_JPEG

*Performs dequantization, inverse DCT and level shift of an 8x8 block.*

---

### Syntax

```
IppStatus ippIDCTQuantInv8x8LS_JPEG_16s8u_C1R(const Ipp16s* pSrc, Ipp8u*  
        pDst,int dstStep, const Ipp16u* pQuantInvTable);
```

### Parameters

<i>pSrc</i>	Pointer to the 8x8 block of quantized DCT coefficients.
<i>pDst</i>	Pointer to the 8x8 block in the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pQuantInvTable</i>	Pointer to the quantization table for the decoder.

### Description

The function `ippIDCTQuantInv8x8LS_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs an inverse DCT, dequantization of DCT coefficients and data conversion from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .

## Level Shift Functions

This section describes the Intel IPP functions that perform level shift operation and are specific for JPEG codec. These functions are listed in [Table 15-5](#).

**Table 15-5**      **Level Shift Functions**

Function Base Name	Description
<a href="#">Sub128_JPEG</a>	Performs level shift to the signed representation.
<a href="#">Add128_JPEG</a>	Performs level shift to the unsigned representation.

The level shift operation is implemented in accordance with its definition in [\[ISO10918\]](#), *Annex A.3.1, Level Shift*.

### Sub128\_JPEG

*Converts data from the unsigned 8u range to the signed 16s range.*

#### Syntax

```
IppStatus ippiSub128_JPEG_8u16s_C1R(const Ipp8u* pSrc, int srcStep,
                                     Ipp16s* pDst, int dstStep, IppiSize roiSize);
```

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiSub128_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts every sample of the source image ROI from the unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> or <code>dstStep</code> has a zero or negative value.

---

## Add128\_JPEG

*Restores samples to the unsigned representation.*

---

## Syntax

```
IppStatus ippiAdd128_JPEG_16s8u_C1R(const Ipp16s* pSrc, int srcStep,  
    Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

## Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.
<code>roiSize</code>	Size of the source and destination ROI in pixels.

## Description

The function `ippiAdd128_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts every sample of source image from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

## Sampling Functions

This section describes Intel IPP functions that perform sampling and are specific for JPEG codec. These functions are listed in [Table 15-6](#).

**Table 15-6 Sampling Functions**

Function Base Name	Description
<a href="#">SampleDownH2V1_JPEG</a>	Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image.
<a href="#">SampleDownH2V2_JPEG</a>	Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image.
<a href="#">SampleDownRowH2V1_Box_JPEG</a>	Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image row.
<a href="#">SampleDownRowH2V2_Box_JPEG</a>	Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image row.
<a href="#">SampleUpH2V1_JPEG</a>	Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image.
<a href="#">SampleUpH2V2_JPEG</a>	Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image.



Table 15-6      Sampling Functions (continued)

Function Base Name	Description
<a href="#"><u>SampleUpRowH2V1_Triangle_JPEG</u></a>	Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image row.
<a href="#"><u>SampleUpRowH2V2_Triangle_JPEG</u></a>	Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image row.
<a href="#"><u>SampleDown444LS_MCU</u></a>	Creates 444 MCU with level shift
<a href="#"><u>SampleDown422LS_MCU</u></a>	Creates 422 MCU with level shift
<a href="#"><u>SampleDown411LS_MCU</u></a>	Creates 411 MCU with level shift
<a href="#"><u>SampleUp444LS_MCU</u></a>	Creates pixel-order image from 444 MCU with level shift
<a href="#"><u>SampleUp422LS_MCU</u></a>	Creates pixel-order image from 422 MCU with level shift
<a href="#"><u>SampleUp411LS_MCU</u></a>	Creates pixel-order image from 411 MCU with level shift

## SampleDownH2V1\_JPEG

*Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image.*

### Syntax

```
IppStatus ippSampleDownH2V1_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,
                                           IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcSize</i>	Size of the source ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstSize</i>	Size of the destination ROI in pixels.

### Description

The function `ippiSampleDownH2V1_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs 2:1 horizontal sampling and 1:1 vertical sampling of an image. Downsampling is performed as a simple “box” filter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value

---

## SampleDownH2V2\_JPEG

*Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image.*

---

### Syntax

```
IppStatus ippiSampleDownH2V2_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,
    IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcSize</i>	Size of the source ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

*dstSize*                      Size of the destination ROI in pixels.

### Description

The function `ippiSampleDownH2V2_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs 2:1 horizontal sampling and 2:1 vertical sampling of an image. Downsampling is performed as a simple “box” filter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

## SampleDownRowH2V1\_Box\_JPEG

*Performs 2:1 horizontal sampling and 1:1 vertical sampling of an image row.*

---

### Syntax

```
IppStatus ippiSampleDownRowH2V1_Box_JPEG_8u_C1(const Ipp8u* pSrc,  
int srcWidth, Ipp8u* pDst);
```

### Parameters

<i>pSrc</i>	Pointer to the source image row.
<i>pDst</i>	Pointer to the destination image row.
<i>srcWidth</i>	Width of the source row in pixels.

### Description

The function `ippiSampleDownRowH2V1_Box_JPEG` is declared in the `ippj.h` file. This function performs 2:1 horizontal sampling and 1:1 vertical sampling of an image row. Downsampling is performed as a simple “box” filter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcWidth</i> has a zero or negative value.

---

## SampleDownRowH2V2\_Box\_JPEG

*Performs 2:1 horizontal sampling and 2:1 vertical sampling of an image row.*

---

### Syntax

```
IppStatus ippiSampleDownRowH2V2_Box_JPEG_8u_C1(const Ipp8u* pSrc1,  
        const Ipp8u* pSrc2, int srcWidth, Ipp8u* pDst);
```

### Parameters

<i>pSrc1</i>	Pointer to the source image row.
<i>pSrc2</i>	Pointer to the subsequent source image row.
<i>pDst</i>	Pointer to the destination image row.
<i>srcWidth</i>	Width of the source row in pixels.

### Description

The function `ippiSampleDownRowH2V2_Box_JPEG` is declared in the `ippj.h` file. This function performs 2:1 horizontal sampling and 2:1 vertical sampling of an image row. Downsampling is performed as a simple “box” filter.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcWidth</i> has a zero or negative value.

---

## SampleUpH2V1\_JPEG

*Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image.*

---

### Syntax

```
IppStatus ippISampleUpH2V1_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,  
    IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

### Parameters

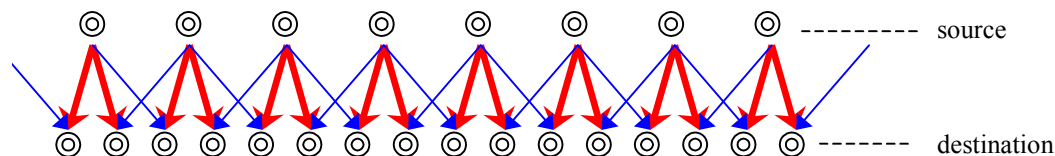
<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcSize</i>	Size of the source ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstSize</i>	Size of the destination ROI in pixels.

### Description

The function `ippISampleUpH2V1_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs 1:2 horizontal sampling and 1:1 vertical sampling of an image. Sampling is performed following the scheme of a “triangle” filter as shown in [Figure 15-1](#) below:

**Figure 15-1 Triangle Sampling Scheme**



Here thick red lines denote source sample weight  $\frac{3}{4}$  and thin blue lines denote source sample weight  $\frac{1}{4}$  in their contribution to the destination sample value.



**NOTE.** This function requires that some samples outside the source image ROI are defined. Specifically, the left-most column must be present and the right-most column may be required.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value

---

## SampleUpH2V2\_JPEG

*Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image.*

---

### Syntax

```
IppStatus ippISampleUpH2V2_JPEG_8u_C1R(const Ipp8u* pSrc, int srcStep,
                                         IppiSize srcSize, Ipp8u* pDst, int dstStep, IppiSize dstSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>srcSize</i>	Size of the source ROI in pixels.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>dstSize</i>	Size of the destination ROI in pixels.

### Description

The function `ippISampleUpH2V2_JPEG` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs 1:2 horizontal sampling and 1:2 vertical sampling of an image. Sampling is performed following the scheme of a “triangle” filter as shown in [Figure 15-1](#).



---

**NOTE.** This function requires that some samples outside the source image ROI are defined. Specifically, the left-most column and the top-most row must be present, whereas the right-most column and the bottom-most row may be required.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> or <i>dstStep</i> has a zero or negative value.

---

## SampleUpRowH2V1\_Triangle\_JPEG

*Performs 1:2 horizontal sampling and 1:1 vertical sampling of an image row.*

---

### Syntax

```
IppStatus ippISampleUpRowH2V1_Triangle_JPEG_8u_C1(const Ipp8u* pSrc,  
int srcWidth, Ipp8u* pDst);
```

### Parameters

<i>pSrc</i>	Pointer to the source image row.
<i>pDst</i>	Pointer to the destination image row.
<i>srcWidth</i>	Width of the source row in pixels.

### Description

The function `ippISampleUpRowH2V1_Triangle_JPEG` is declared in the `ippj.h` file. This function performs 1:2 horizontal sampling and 1:1 vertical sampling of an image row. Sampling is performed following the scheme of a “triangle” filter as shown in [Figure 15-1](#). This function does not require samples outside the source image row.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcWidth</i> has a zero or negative value.



---

## SampleUpRowH2V2\_Triangle\_JPEG

*Performs 1:2 horizontal sampling and 1:2 vertical sampling of an image row.*

---

### Syntax

```
IppStatus ippiSampleUpRowH2V2_Triangle_JPEG_8u_C1(const Ipp8u* pSrc1,  
    const Ipp8u* pSrc2, int srcWidth, Ipp8u* pDst);
```

### Parameters

<i>pSrc1</i>	Pointer to the source image row.
<i>pSrc2</i>	Pointer to the subsequent source image row.
<i>pDst</i>	Pointer to the destination image row.
<i>srcWidth</i>	Width of the source row in pixels.

### Description

The function `ippiSampleUpRowH2V2_Triangle_JPEG` is declared in the `ippj.h` file. This function performs 1:2 horizontal sampling and 1:2 vertical sampling of an image row. Sampling is performed following the scheme of a “triangle” filter as shown in [Figure 15-1](#). This function does not require samples outside the source image rows.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all specified pointers are NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcWidth</i> has a zero or negative value.

---

## SampleDown444LS\_MCU

*Creates 444 MCU with level shift from pixel-order data.*

---

### Syntax

```
IppStatus ippiSampleDown444LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc,  
        int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Array of 3 pointers to the destination image blocks.

### Description

The function `ippiSampleDown444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts interleaved data to the 444 MCU that consists of three 8x8 blocks - one block for each channel (see [Figure 6-13](#)).

Additionally, this function converts data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers are NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

---

## SampleDown422LS\_MCU

*Creates 422 MCU with level shift from pixel-order data.*

---

### Syntax

```
IppStatus ippISampleDown422LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc,  
        int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Pointer to the array of 3 pointers to the destination blocks.

### Description

The function `ippISampleDown422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts interleaved data to the full MCU with 4:2:2 downsampling. Thus, final MCU consists of four blocks - two 8x8 blocks for channel 1, one 8x8 block for channel 2, and one 8x8 block for channel 3 (see [Figure 6-13](#)).

Downsampling is performed by averaging the corresponding pixel values.

Additionally, this function converts data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers are <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

---

## SampleDown411LS\_MCU

*Creates 411 MCU with level shift from pixel-order data.*

---

### Syntax

```
IppStatus ippiSampleDown411LS_MCU_8u16s_C3P3R(const Ipp8u* pSrc,  
        int srcStep, Ipp16s* pDstMCU[3]);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Pointer to the array of 3 pointers to the destination blocks.

### Description

The function `ippiSampleDown411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function converts interleaved data to the full MCU with 4:1:1 downsampling. Thus, final MCU consists of six blocks - four 8x8 blocks for channel 1, one 8x8 block for channel 2, and one 8x8 block for channel 3 (see [Figure 6-13](#)).

Downsampling is performed by averaging the corresponding pixel values.

Additionally, this function converts data from unsigned `Ipp8u` range [0..255] to the signed `Ipp16s` range [-128..127] performing level shift operation (by subtracting 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers are NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> has a zero or negative value.

---

## SampleUp444LS\_MCU

*Creates pixel-order image from 444 MCU and performs level shift.*

---

### Syntax

```
IppStatus ippiSampleUp444LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
      Ipp8u* pDst, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Pointer to the array of 3 pointers to the source blocks.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiSampleUp444LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates pixel-order image from 444 MCU (see [Figure 6-13](#)).

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

---

## SampleUp422LS\_MCU

*Creates pixel-order image from 422 MCU and performs level shift.*

---

### Syntax

```
IppStatus ippiSampleUp422LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
      Ipp8u* pDst, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Pointer to the array of 3 pointers to the source blocks.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiSampleUp422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates pixel-order image from 422 MCU (see [Figure 6-13](#)). Upsampling is performed as a simple box filter.

Additionally, this function converts data from the signed `Ipp16s` range `[-128 . . 127]` to the unsigned `Ipp8u` range `[0 . . 255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

---

## SampleUp411LS\_MCU

*Creates pixel-order image from 411 MCU and performs level shift.*

---

### Syntax

```
IppStatus ippiSampleUp411LS_MCU_16s8u_P3C3R(const Ipp16s* pSrcMCU[3],  
      Ipp8u* pDst, int dstStep);
```

### Parameters

<i>pSrcMCU</i>	Pointer to the array of 3 pointers to the source blocks.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiSampleUp411LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates pixel-order image from 411 MCU (see [Figure 6-13](#)). Upsampling is performed as a simple box filter.

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has a zero or negative value.

## Planar-to-Pixel and Pixel-to-Planar Conversion Functions

This section describes the functions that convert planar (not-interleaved) data to pixel-order (interleaved) data and vice versa. [Table 15-7](#) lists these functions described in more detail later in this section.

**Table 15-7 Pixel-to-Planar and Planar-to-Pixel Conversion Functions**

Function Base Name	Description
<a href="#">Split422LS_MCU</a>	Creates 422 MCU from 422 interleaved data with level shift.
<a href="#">Join422LS_MCU</a>	Creates 422 interleaved data from 422 MCU with level shift.

---

### Split422LS\_MCU

*Creates 422 MCU from 422 interleaved data with level shift.*

---

#### Syntax

```
IppStatus ippiSplit422LS_MCU_8u16s_C2P3R(const Ipp8u* pSrc, int SrcStep,
    Ipp16s* pDstMCU[3]);
```

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstMCU</i>	Pointer to the array of 3 pointers to the destination blocks.

#### Description

The function `ippiSplit422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates 422 MCU from 422 pixel-order data (see [Figure 6-15](#)), that is, converts pixel-order data to planar form without re-sampling. Final MCU consists of four blocks - two 8x8 blocks for channel 1, one 8x8 block for channel 2, and one 8x8 block for channel 3 (see [Figure 6-13](#)).



Additionally, this function converts data from unsigned `Ipp8u` range `[0..255]` to the signed `Ipp16s` range `[-128..127]` performing level shift operation (by subtracting 128).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers are <code>NULL</code> .
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has a zero or negative value.

---

## Join422LS\_MCU

*Creates 422 interleaved data from 422 MCU with level shift.*

---

### Syntax

```
IppStatus ippiJoin422LS_MCU_16s8u_P3C2R(const Ipp16s* pSrcMCU[3], Ipp8u*
    pDst, int dstStep);
```

### Parameters

<code>pSrcMCU</code>	Pointer to the array of 3 pointers to the source blocks.
<code>pDst</code>	Pointer to the destination image ROI.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the destination image.

### Description

The function `ippiJoin422LS_MCU` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function creates 422 interleaved data (see [Table 6-2](#)) from 422 MCU (see [Figure 6-13](#)), that is, converts planar data to pixel-order form without re-sampling. Output data is a repetitive sequence of pixels `Ch1Ch2Ch1Ch3...`

Additionally, this function converts data from the signed `Ipp16s` range `[-128..127]` to the unsigned `Ipp8u` range `[0..255]` performing level shift operation (by adding 128).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers are NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <code>dstStep</code> has a zero or negative value.

## Huffman Codec Functions

This section describes Huffman encoding and decoding functions that are specific for JPEG codec. These functions are listed in [Table 15-8](#).

**Table 15-8 Huffman Codec Functions**

Function Base Name	Description
<b>Encoder Functions</b>	
<a href="#"><code>EncodeHuffmanRawTableInit_JPEG</code></a>	Creates raw Huffman tables using Huffman symbols statistics.
<a href="#"><code>EncodeHuffmanSpecGetBufSize_JPEG</code></a>	Returns the length of the <code>IppiEncodeHuffmanSpec</code> structure.
<a href="#"><code>EncodeHuffmanSpecInit_JPEG</code></a>	Creates Huffman table in format that is suitable for an encoder.
<a href="#"><code>EncodeHuffmanSpecInitAlloc_JPEG</code></a>	Allocates memory and creates Huffman table in format that is suitable for an encoder.
<a href="#"><code>EncodeHuffmanSpecFree_JPEG</code></a>	Frees memory allocated by the <code>ippiEncodeHuffmanSpecInitAlloc_JPEG_8u</code> function.
<a href="#"><code>EncodeHuffmanStateGetBufSize_JPEG</code></a>	Returns the length of the <code>IppiEncodeHuffmanState</code> structure.
<a href="#"><code>EncodeHuffmanStateInit_JPEG</code></a>	Initializes the <code>IppiEncodeHuffmanState</code> structure.
<a href="#"><code>EncodeHuffmanStateInitAlloc_JPEG</code></a>	Allocates memory and initializes the <code>IppiEncodeHuffmanState</code> structure.
<a href="#"><code>EncodeHuffmanStateFree_JPEG</code></a>	Frees memory allocated by the <code>ippiEncodeHuffmanStateInitAlloc_JPEG</code> function.
<a href="#"><code>EncodeHuffman8x8_JPEG</code></a>	Performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.
<a href="#"><code>EncodeHuffman8x8_Direct_JPEG</code></a>	Directly performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.

**Table 15-8 Huffman Codec Functions (continued)**

<a href="#"><u>GetHuffmanStatistics8x8_JPEG</u></a>	Computes Huffman symbols statistics for the baseline encoding.
<a href="#"><u>GetHuffmanStatistics8x8_DCFirst_JPEG</u></a>	Computes Huffman symbols statistics for the progressive encoding (DC coefficient).
<a href="#"><u>GetHuffmanStatistics8x8_ACFirst_JPEG</u></a>	Computes Huffman symbols statistics for the progressive encoding (AC coefficients, the first scan).
<a href="#"><u>GetHuffmanStatistics8x8_ACRefine_JPEG</u></a>	Computes Huffman symbols statistics for the progressive encoding (AC coefficients, subsequent scans).
<a href="#"><u>EncodeHuffman8x8_DCFirst_JPEG</u></a>	Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).
<a href="#"><u>EncodeHuffman8x8_DCRefine_JPEG</u></a>	Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (subsequent scans).
<a href="#"><u>EncodeHuffman8x8_ACFirst_JPEG</u></a>	Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).
<a href="#"><u>EncodeHuffman8x8_ACRefine_JPEG</u></a>	Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (subsequent scans).
<b>Decoder Functions</b>	
<a href="#"><u>DecodeHuffmanSpecGetBufSize_JPEG</u></a>	Returns the length of the <code>IppiDecodeHuffmanSpec</code> structure.
<a href="#"><u>DecodeHuffmanSpecInit_JPEG</u></a>	Creates Huffman table in format that is suitable for a decoder.
<a href="#"><u>DecodeHuffmanSpecInitAlloc_JPEG</u></a>	Allocates memory and creates Huffman table in format that is suitable for a decoder.
<a href="#"><u>DecodeHuffmanSpecFree_JPEG</u></a>	Frees memory allocated by the <code>ippiDecodeHuffmanSpecInitAlloc_JPEG</code> function.
<a href="#"><u>DecodeHuffmanStateGetBufSize_JPEG</u></a>	Returns the length of the <code>IppiDecodeHuffmanState</code> structure
<a href="#"><u>DecodeHuffmanStateInit_JPEG</u></a>	Initializes the <code>IppiDecodeHuffmanState</code> structure.
<a href="#"><u>DecodeHuffmanStateInitAlloc_JPEG</u></a>	Allocates memory and initializes the <code>IppiDecodeHuffmanState</code> structure.

**Table 15-8**      **Huffman Codec Functions (continued)**

<a href="#">DecodeHuffmanStateFree_JPEG</a>	Frees memory allocated by the <code>ippiDecodeHuffmanStateInitAlloc_JPEG</code> function.
<a href="#">DecodeHuffman8x8_JPEG</a>	Performs Huffman baseline decoding of 8x8 block of the quantized DCT coefficients.
<a href="#">DecodeHuffman8x8_Direct_JPEG</a>	Directly performs Huffman baseline decoding of 8x8 block of the quantized DCT coefficients.
<a href="#">DecodeHuffman8x8_DCFirst_JPEG</a>	Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).
<a href="#">DecodeHuffman8x8_DCRefine_JPEG</a>	Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (subsequent scans).
<a href="#">DecodeHuffman8x8_ACFirst_JPEG</a>	Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).
<a href="#">DecodeHuffman8x8_ACRefine_JPEG</a>	Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (subsequent scans).

## EncodeHuffmanRawTableInit\_JPEG

*Creates raw Huffman tables using Huffman symbols statistics.*

### Syntax

```
IppStatusippiEncodeHuffmanRawTableInit_JPEG_8u(constintpStatistics[256],
        Ipp8u* pListBits, Ipp8u* pListVals);
```

### Parameters

<i>pStatistics</i>	Pointer to the buffer that contains Huffman symbol statistics.
<i>pListBits</i>	Pointer to the <code>Bits</code> list.
<i>pListVals</i>	Pointer to the <code>Vals</code> list.

## Description

The function `ippiEncodeHuffmanRawTableInit_JPEG` is declared in the `ippj.h` file. This function builds raw Huffman tables using the previously computed Huffman symbols statistics. An array for statistics is indexed with Huffman symbol value. The values in the array are frequencies of a given symbol.

The `Bits` and `Vals` lists are specified in [\[ISO10918\]](#), Annex B, Figure B.7.

To generate `Bits` and `Vals` lists, the function uses the procedure that is described in [\[ISO10918\]](#), Annex K.2, *A Procedure for Generating the Lists, Which Specify a Huffman Code Table*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all specified pointers are <code>NULL</code> .

---

## EncodeHuffmanSpecGetBufSize\_JPEG

*Returns the length of the `IppiEncodeHuffmanSpec` structure.*

---

## Syntax

```
IppStatus ippiEncodeHuffmanSpecGetBufSize_JPEG_8u(int* pSize);
```

## Parameters

<code>pSize</code>	Pointer to a variable that will receive the length of the <code>IppiEncodeHuffmanSpec</code> structure.
--------------------	---

## Description

The function `ippiEncodeHuffmanSpecGetBufSize_JPEG` is declared in the `ippj.h` file. This function computes the length of the `IppiEncodeHuffmanSpec` structure.




---

**NOTE.** The fields of the structure `IppiEncodeHuffmanSpec` are hidden from the user.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer is NULL.

---

## EncodeHuffmanSpecInit\_JPEG

*Creates Huffman table in a format that is suitable for an encoder.*

---

### Syntax

```
IppStatus ippEncodeHuffmanSpecInit_JPEG_8u(const Ipp8u* pListBits,  
      const Ipp8u* pListVals, IppiEncodeHuffmanSpec* pEncHuffSpec);
```

### Parameters

<code>pListBits</code>	Pointer to Bits list.
<code>pListVals</code>	Pointer to Vals list.
<code>pEncHuffSpec</code>	Pointer to the Huffman table for the encoder.

### Description

The function `ippEncodeHuffmanSpecInit_JPEG` is declared in the `ippj.h` file. This function creates Huffman table in a format that is suitable for the encoder.

The Bits and Vals lists are specified in [\[ISO10918\]](#), Annex C, *Huffman table specification*. To generate the table, the function uses the procedure that is described in [\[ISO10918\]](#), Annex C.2, *Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all specified pointers are NULL.
<code>ippStsJPEGHuffTableErr</code>	Indicates an error condition during the initialization of the table. It may happen when raw tables contain inadmissible values.

---

## EncodeHuffmanSpecInitAlloc\_JPEG

*Allocates memory and creates Huffman table in a format that is suitable for an encoder.*

---

### Syntax

```
IppStatus ippiEncodeHuffmanSpecInitAlloc_JPEG_8u(const Ipp8u* pListBits,
          const Ipp8u* pListVals, IppiEncodeHuffmanSpec** ppEncHuffSpec);
```

### Parameters

<i>pListBits</i>	Pointer to the Bits list.
<i>pListVals</i>	Pointer to the Vals list.
<i>ppEncHuffSpec</i>	Pointer to the returned pointer to the Huffman table for the coder.

### Description

The function `ippiEncodeHuffmanSpecInitAlloc_JPEG` is declared in the `ippj.h` file. This function allocates memory and creates Huffman table in a format that is suitable for the encoder. The Bits and Vals lists are specified in [\[ISO10918\]](#), Annex C, *Huffman table specification*. To generate the table, the function uses the procedure that is described in [\[ISO10918\]](#), Annex C.2, *Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all specified pointers are NULL.
<code>ippStsJPEGHuffTableErr</code>	Indicates an error condition when the table is initialized. It may happen when raw tables contain inadmissible values.

---

## EncodeHuffmanSpecFree\_JPEG

*Frees memory allocated by the*

`ippiEncodeHuffmanSpecInitAlloc_JPEG_8u`  
*function.*

---

### Syntax

```
IppStatus ippiEncodeHuffmanSpecFree_JPEG_8u(  
    IppiEncodeHuffmanSpec* pEncHuffSpec);
```

### Parameters

*pEncHuffSpec*                      Pointer to the Huffman table for the encoder.

### Description

The function `ippiEncodeHuffmanSpecFree_JPEG` is declared in the `ippj.h` file. This function frees memory allocated by the `ippiEncodeHuffmanSpecInitAlloc_JPEG` function for the Huffman table.

### Return Values

`ippStsNoErr`                      Indicates no error.

---

## EncodeHuffmanStateGetBufSize\_JPEG

*Returns the length of the `IppiEncodeHuffmanState` structure.*

---

```
IppStatus ippiEncodeHuffmanStateGetBufSize_JPEG_8u(int* pSize);
```

### Parameters

*pSize*                              Pointer to a variable that will receive the length of the `IppiEncodeHuffmanState` structure.



## Description

The function `ippiEncodeHuffmanStateGetBufSize_JPEG` is declared in the `ippj.h` file. This function computes the length (in bytes) of the `IppiEncodeHuffmanState` structure.



---

**NOTE.** The fields of the structure `IppiEncodeHuffmanState` are hidden from the user.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if specified pointer is NULL.

---

## EncodeHuffmanStateInit\_JPEG

*Initializes the `IppiEncodeHuffmanState` structure.*

---

## Syntax

```
IppStatus ippiEncodeHuffmanStateInit_JPEG_8u( IppiEncodeHuffmanState*  
        pEncHuffState);
```

## Parameters

<i>pEncHuffState</i>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
----------------------	---

## Description

The function `ippiEncodeHuffmanStateInit_JPEG` is declared in the `ippj.h` file. This function initializes the `IppiEncodeHuffmanState` to the initial state. This function must be called prior to the JPEG data stream encoding.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.

---

## EncodeHuffmanStateInitAlloc\_JPEG

*Allocates memory and initializes the  
IppiEncodeHuffmanState structure.*

---

### Syntax

```
IppStatus ippiEncodeHuffmanStateInitAlloc_JPEG_8u(IppiEncodeHuffmanState**  
    ppEncHuffState);
```

### Parameters

*ppEncHuffState*      Pointer to the IppiEncodeHuffmanState structure.

### Description

The function ippiEncodeHuffmanStateInitAlloc\_JPEG is declared in the `ippj.h` file. This function allocates memory and initializes the IppiEncodeHuffmanState structure to the initial state. This function must be called prior to the JPEG data stream encoding.

### Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one or all of the specified pointers are NULL.

---

## EncodeHuffmanStateFree\_JPEG

*Frees memory allocated by the  
ippiEncodeHuffmanStateInitAlloc\_JPEG  
function.*

---

### Syntax

```
IppStatus ippiEncodeHuffmanStateFree_JPEG_8u(IppiEncodeHuffmanState*  
    pEncHuffState);
```

*pEncHuffState*      Pointer to the IppiEncodeHuffmanState structure.

## Description

The function `ippiEncodeHuffmanStateFree_JPEG` is declared in the `ippj.h` file. This function frees memory allocated for the `IppiEncodeHuffmanState` structure.

## Return Values

`ippStsNoErr` Indicates no error.

## EncodeHuffman8x8\_JPEG

*Performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.*

## Syntax

```
IppStatus ippiEncodeHuffman8x8_JPEG_16s1u_C1(const Ipp16s* pSrc,
    Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, Ipp16s* pLastDC,
    const IppiEncodeHuffmanSpec* pDcTable, const IppiEncodeHuffmanSpec*
    pAcTable, IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

## Parameters

<code>pSrc</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pDst</code>	Pointer to the buffer for the output bit stream.
<code>dstLenBytes</code>	Length in bytes of the buffer for the bit stream.
<code>pDstCurrPos</code>	Shift in bytes of the current byte in the output buffer.
<code>pLastDC</code>	Pointer to the DC coefficient of the previous 8x8 block.
<code>pDcTable</code>	Pointer to the Huffman table for the DC coefficients.
<code>pAcTable</code>	Pointer to the Huffman table for the AC coefficients.
<code>pEncHuffState</code>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
<code>bFlushState</code>	Set to 1 for the last 8x8 block in the scan.

## Description

The function `ippiEncodeHuffman8x8_JPEG` is declared in the `ippj.h` file. This function performs encoding of an 8x8 block of the quantized DCT coefficients using `pDcTable` and `pAcTable` tables. Encoding procedure is specified in [ISO10918], *Annex F.1.2, Baseline Huffman Encoding Procedures*.

Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits, which do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter `bFlushState` to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <code>dstLenBytes</code> has a zero or negative value, or <code>pDstCurrPos</code> is outside the limit of <code>dstLenBytes</code> .
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if a DCT coefficient is out of the allowed range $(-1023..1023)$ .

---

## EncodeHuffman8x8\_Direct\_JPEG

*Directly performs Huffman baseline encoding of an 8x8 block of quantized DCT coefficients.*

---

## Syntax

```
IppStatus ippiEncodeHuffman8x8_Direct_JPEG_16s1u_C1(const Ipp16s* pSrc,  
    Ipp8u* pDst, int* pDstBitsLen, Ipp16s* pLastDC,  
    const IppiEncodeHuffmanSpec* pDcTable,  
    const IppiEncodeHuffmanSpec* pAcTable);
```

## Parameters

<code>pSrc</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pDst</code>	Pointer to the buffer for the output bit stream.

<i>pDstBitsLen</i>	Length in bits of the buffer for the bit stream.
<i>pLastDC</i>	Pointer to the DC coefficient of the previous 8x8 block of the same color component.
<i>pDcTable</i>	Pointer to the Huffman table for the DC coefficients.
<i>pAcTable</i>	Pointer to the Huffman table for the AC coefficients.

### Description

The function `ippiEncodeHuffman8x8_Direct_JPEG` is declared in the `ippj.h` file. This function directly performs encoding of an 8x8 block of the quantized DCT coefficients using *pDcTable* and *pAcTable* tables. The function does not require any additional structure for operation.

Encoding procedure is specified in [\[ISO10918\]](#), *Annex F.1.2, Baseline Huffman Encoding Procedures*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.

---

## GetHuffmanStatistics8x8\_JPEG

*Computes Huffman symbols statistics for the baseline encoding.*

---

### Syntax

```
IppStatus ippiGetHuffmanStatistics8x8_JPEG_16s_C1(const Ipp16s* pSrc,  
int pDcStatistics[256], int pAcStatistics[256], Ipp16s* pLastDC);
```

### Parameters

<i>pSrc</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pDcStatistics</i>	Pointer to the Huffman symbols statistics buffer for the DC coefficients.
<i>pAcStatistics</i>	Pointer to the Huffman symbols statistics buffer for the AC coefficients.
<i>pLastDC</i>	Pointer to the DC coefficient of the previous 8x8 block.

### Description

The function `ippiGetHuffmanStatistics8x8_JPEG` is declared in the `ippj.h` file. This function computes the statistics of Huffman symbols for an 8x8 block of the quantized DCT coefficients for the baseline encoding.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if a DCT coefficient is out of the allowed range $(-1023..1023)$ .

---

## GetHuffmanStatistics8x8\_DCFirst\_JPEG

*Computes Huffman symbols statistics for the progressive encoding (DC coefficients).*

---

### Syntax

```
IppStatus ippiGetHuffmanStatistics8x8_DCFirst_JPEG_16s_C1(const Ipp16s* pSrc,  
    int pDcStatistics[256], Ipp16s* pLastDC, int Al);
```

### Parameters

<i>pSrc</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pDcStatistics</i>	Pointer to the Huffman symbols statistics buffer for the DC coefficients.
<i>pLastDC</i>	Pointer to the DC coefficient of the previous 8x8 block.
<i>Al</i>	Successive approximation bit position low, it specifies the actual point transform.

### Description

The function `ippiGetHuffmanStatistics8x8_DCFirst_JPEG` is declared in the `ippj.h` file. This function computes the Huffman symbols statistics for an 8x8 block of the quantized DCT coefficients for progressive encoding, the first scan, DC coefficients.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or both of the specified pointers are NULL.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if a DCT coefficient is out of the allowed range $(-1023..1023)$ .

---

## GetHuffmanStatistics8x8\_ACFirst\_JPEG

*Computes Huffman symbols statistics for the progressive encoding (AC coefficients, the first scan).*

---

### Syntax

```
IppStatus ippGetHuffmanStatistics8x8_ACFirst_JPEG_16s_C1(const Ipp16s* pSrc,  
int pAcStatistics[256], int Ss, int Se, int Al, IppiEncodeHuffmanState*  
pEncHuffState, int bFlushState);
```

### Parameters

<code>pSrc</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pAcStatistics</code>	Pointer to the Huffman symbols statistics buffer for the AC coefficients.
<code>Ss</code>	Spectral selection start index.
<code>Se</code>	Spectral selection end index.
<code>Al</code>	Successive approximation bit positions low, it specifies the actual point transform.
<code>pEncHuffState</code>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
<code>bFlushState</code>	Set to 1 for the last 8x8 block in the scan.

### Description

The function `ippGetHuffmanStatistics8x8_ACFirst_JPEG` is declared in the `ippj.h` file. This function computes the Huffman symbols statistics for an 8x8 block of the quantized DCT coefficients for progressive encoding, the first scan, AC coefficients.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if a DCT coefficient is out of the allowed range $(-1023..1023)$ .

---

## GetHuffmanStatistics8x8\_ACRefine\_JPEG

*Computes Huffman symbols statistics for the progressive encoding (AC coefficients, subsequent scans).*

---

### Syntax

```
IppStatus ippGetHuffmanStatistics8x8_ACRefine_JPEG_16s_C1(const Ipp16s* pSrc,  
    int pAcStatistics[256], int Ss, int Se, int Al, IppiEncodeHuffmanState*  
    pEncHuffState, int bFlushState);
```

### Parameters

<code>pSrc</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pAcStatistics</code>	Pointer to the Huffman symbols statistics buffer for the AC coefficients.
<code>Ss</code>	Spectral selection start index.
<code>Se</code>	Spectral selection end index.
<code>Al</code>	Successive approximation bit positions low, it specifies the actual point transform.
<code>pEncHuffState</code>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
<code>bFlushState</code>	Set to 1 for the last 8x8 block in the scan.

### Description

The function `ippGetHuffmanStatistics8x8_ACRefine_JPEG` is declared in the `ippj.h` file. This function computes the Huffman symbols statistics for an 8x8 block of the quantized DCT coefficients for progressive encoding, the subsequent scans, AC coefficients.



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if a DCT coefficient is out of the allowed range (-1023..1023).

---

## EncodeHuffman8x8\_DCFirst\_JPEG

*Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).*

---

### Syntax

```
IppStatus ippEncodeHuffman8x8_DCFirst_JPEG_16slu_C1(const Ipp16s* pSrc, Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, Ipp16s* pLastDC, int Al, IppiEncodeHuffmanSpec* pDcTable, IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

### Parameters

<code>pSrc</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pDst</code>	Pointer to the buffer for the output bit stream.
<code>dstLenBytes</code>	Length in bytes of the buffer for the bit stream.
<code>pDstCurrPos</code>	Shift in bytes of the current byte in the output buffer.
<code>pLastDC</code>	Pointer to the DC coefficient of the previous 8x8 block.
<code>Al</code>	Successive approximation bit positions low, it specifies the actual point transform.
<code>pDcTable</code>	Pointer to the Huffman table for DC coefficients.
<code>pEncHuffState</code>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
<code>bFlushState</code>	Set it to 1 for the last 8x8 block in the scan

## Description

The function `ippiEncodeHuffman8x8_DCFirst_JPEG` is declared in the `ippj.h` file. This function performs encoding of the DC coefficient from an 8x8 block of the quantized coefficients (progressive encoding, the first scan), using `pDcTable` table. The encoding procedure conforms to [\[ISO10918\]](#), *Annex G.1.2, Progressive Encoding Procedures with Huffman*.

Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits, which do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter `bFlushState` to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023..1023)$ .

---

## EncodeHuffman8x8\_DCRefine\_JPEG

*Performs progressive encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (subsequent scans).*

---

## Syntax

```
IppStatus ippiEncodeHuffman8x8_DCRefine_JPEG_16s1u_C1(const Ipp16s* pSrc,  
    Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, int Al,  
    IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

## Parameters

<code>pSrc</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pDst</code>	Pointer to the buffer for the output bit stream.
<code>dstLenBytes</code>	Length in bytes of the buffer for the bit stream.

<i>pDstCurrPos</i>	Shift in bytes of the current byte in the output buffer.
<i>Al</i>	Successive approximation bit positions low, it specifies the actual point transform.
<i>pEncHuffState</i>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
<i>bFlushState</i>	Set it to 1 for the last 8x8 block in the scan

## Description

The `ippiEncodeHuffman8x8_DCRefine_JPEG` is declared in the `ippj.h` file. This function performs encoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (progressive encoding, the subsequent scans), using *pDcTable* table.

The encoding procedure conforms to [\[ISO10918\]](#), *Annex G.1.2, Progressive Encoding Procedures with Huffman*.

Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits, which do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter *bFlushState* to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023 \dots 1023)$ .

---

## EncodeHuffman8x8\_ACFirst\_JPEG

*Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).*

---

### Syntax

```
IppStatus ippEncodeHuffman8x8_ACFirst_JPEG_16s1u_C1(const Ipp16s* pSrc, Ipp8u*  
    pDst, int dstLenBytes, int* pDstCurrPos, int Ss, int Se, int Al,  
    IppiEncodeHuffmanSpec* pAcTable, IppiEncodeHuffmanState* pEncHuffState, int  
    bFlushState);
```

### Parameters

<i>pSrc</i>	Pointer to the 8x8 block of the quantized DCT coefficients
<i>pDst</i>	Pointer to the buffer for the output bit stream.
<i>dstLenBytes</i>	Length in bytes of the buffer for the bit stream.
<i>pDstCurrPos</i>	Shift in bytes of the current byte in the output buffer.
<i>Ss</i>	Spectral selection start index.
<i>Se</i>	Spectral selection end index.
<i>Al</i>	Successive approximation bit positions low, it specifies the actual point transform.
<i>pAcTable</i>	Pointer to the Huffman table for AC coefficients.
<i>pEncHuffState</i>	Pointer to the IppiEncodeHuffmanState structure.
<i>bFlushState</i>	Set it to 1 for the last 8x8 block in the scan.

### Description

The function `ippEncodeHuffman8x8_ACFirst_JPEG` is declared in the `ippj.h` file. This function performs encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (progressive encoding, the first scan), using *pAcTable* table.

The encoding procedure conforms to [\[ISO10918\]](#), Annex G.1.2, *Progressive Encoding Procedures with Huffman*.

Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure

collects the bits, which do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter `bFlushState` to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023 \dots 1023)$ .

---

## EncodeHuffman8x8\_ACRefine\_JPEG

*Performs progressive encoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients.*

---

### Syntax

```
IppStatus ippiEncodeHuffman8x8_ACRefine_JPEG_16s1u_C1(const Ipp16s* pSrc,
    Ipp8u* pDst, int dstLenBytes, int* pDstCurrPos, int Ss, int Se, int Al,
    IppiEncodeHuffmanSpec* pAcTable, IppiEncodeHuffmanState* pEncHuffState,
    int bFlushState);
```

### Parameters

<code>pSrc</code>	Pointer to the 8x8 block of the quantized DCT coefficients.
<code>pDst</code>	Pointer to the buffer for the output bit stream.
<code>dstLenBytes</code>	Length in bytes of the buffer for the bit stream.
<code>pDstCurrPos</code>	Shift in bytes of the current byte in the output buffer.
<code>Ss</code>	Spectral selection start index.
<code>Se</code>	Spectral selection end index.
<code>Al</code>	Successive approximation bit positions low, it specifies the actual point transform.

<i>pAcTable</i>	Pointer to the Huffman table for AC coefficients.
<i>pEncHuffState</i>	Pointer to the <code>IppiEncodeHuffmanState</code> structure.
<i>bFlushState</i>	Set it to 1 for the last 8x8 block in the scan.

### Description

The function `ippiEncodeHuffman8x8_ACRefine_JPEG` is declared in the `ippj.h` file. This function performs encoding of the AC coefficient from an 8x8 block of the quantized DCT coefficients (progressive encoding, the subsequent scans), using *pAcTable* table. The encoding procedure conforms to [\[ISO10918\]](#), *Annex G.1.2, Progressive Encoding Procedures with Huffman*.

Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits, which do not make up a complete byte. To force adding the bits accumulated in `IppiEncodeHuffmanState` to the output buffer, set the parameter *bFlushState* to 1 for the last 8x8 block in the scan or restart interval being encoded. In all other cases it must be set to zero.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023..1023)$ .

---

## DecodeHuffmanSpecGetBufSize\_JPEG

*Returns the length of the `IppiDecodeHuffmanSpec` structure.*

---

### Syntax

```
IppStatus ippiDecodeHuffmanSpecGetBufSize_JPEG_8u(int* pSize);
```

## Parameters

*pSize* Pointer to a variable that will receive the length of the `IppiDecodeHuffmanSpec` structure.

## Description

The function `ippiDecodeHuffmanSpecGetBufSize_JPEG` is declared in the `ippj.h` file. This function returns the length of the `IppiDecodeHuffmanSpec` structure.



---

**NOTE.** The fields of the structure `IppiDecodeHuffmanSpec` are hidden from the user.

---

## Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error condition if the pointer is `NULL`.

---

## DecodeHuffmanSpecInit\_JPEG

*Creates Huffman table in a format  
that is suitable for a decoder*

---

## Syntax

```
ippiDecodeHuffmanSpecInit_JPEG_8u(const Ipp8u* pListBits,  
    const Ipp8u* pListVals, IppiDecodeHuffmanSpec* pDecHuffSpec);
```

## Parameters

*pListBits* Pointer to Bits list.

*pListVals* Pointer to Vals list.

*pDecHuffSpec* Pointer to the Huffman table for the decoder.

### Description

The function `ippiDecodeHuffmanSpecInit_JPEG` is declared in the `ippj.h` file. This function creates Huffman table in a format that is suitable for a decoder.

The `Bits` and `Vals` lists are specified in [\[ISO10918\]](#), *Annex C, Huffman Table Specification*. To generate the table, the function uses the procedure that is described in [\[ISO10918\]](#), *Annex C.2, Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all specified pointers are <code>NULL</code> .
<code>ippStsJPEGHuffTableErr</code>	Indicates an error condition during the initialization of the table. It may happen when raw tables contain inadmissible values.

---

## DecodeHuffmanSpecInitAlloc\_JPEG

*Allocates memory and creates Huffman table in a format that is suitable for a decoder.*

---

### Syntax

```
IppStatus ippiDecodeHuffmanSpecInitAlloc_JPEG_8u(const Ipp8u* pListBits,  
          const Ipp8u* pListVals, IppiDecodeHuffmanSpec** ppDecHuffSpec);
```

### Parameters

<code>pListBits</code>	Pointer to <code>Bits</code> list.
<code>pListVals</code>	Pointer to <code>Vals</code> list.
<code>ppDecHuffSpec</code>	Pointer to the returned pointer to the Huffman table for the decoder.

### Description

The function `ippiDecodeHuffmanSpecInitAlloc_JPEG` is declared in the `ippj.h` file. This function allocates memory and creates the Huffman table in a format that is suitable for a decoder. The function expects that `Bits` and `Vals` lists are supplied in the format specified in [\[ISO10918\]](#),



*Annex C, Huffman Table Specification.*

To generate the table, the function uses the procedure that is described in [\[ISO10918\]](#), *Annex C.2, Conversion of Huffman Tables Specified in Interchange Format to Tables of Codes and Code Lengths*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGHuffTableErr</code>	Indicates an error condition during the initialization of the table. It may happen when raw tables contain inadmissible values.

---

## DecodeHuffmanSpecFree\_JPEG

*Frees memory allocated by*

`ippiDecodeHuffmanSpecInitAlloc_JPEG`  
*function.*

---

### Syntax

```
IppStatus ippiDecodeHuffmanSpecFree_JPEG_8u(IppiDecodeHuffmanSpec*  
    pDecHuffSpec);
```

### Parameters

*pDecHuffSpec*                      Pointer to the Huffman table for the decoder.

### Description

The function `ippiDecodeHuffmanSpecFree_JPEG` is declared in the `ippj.h` file. This function frees memory allocated by the `ippiDecodeHuffmanSpecInitAlloc_JPEG` function for the Huffman table.

### Return Values

`ippStsNoErr`                      Indicates no error.

---

## DecodeHuffmanStateGetBufSize\_JPEG

Returns the length of `IppiDecodeHuffmanState` structure.

---

### Syntax

```
IppStatus ippiDecodeHuffmanStateGetBufSize_JPEG_8u(int* pSize);
```

### Parameters

<code>pSize</code>	Pointer to a variable that will receive the length of the <code>IppiDecodeHuffmanState</code> structure.
--------------------	--

### Description

The function `ippiDecodeHuffmanStateGetBufSize_JPEG` is declared in the `ippj.h` file. This function returns the length in bytes of the `IppiDecodeHuffmanState` structure.



---

**NOTE.** The fields of the structure `IppiDecodeHuffmanState` are hidden from the user.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the specified pointer is <code>NULL</code> .

---

## DecodeHuffmanStateInit\_JPEG

Initializes `IppiDecodeHuffmanState` structure.

---

### Syntax

```
IppStatus ippiDecodeHuffmanStateInit_JPEG_8u(IppiDecodeHuffmanState*  
pDecHuffState);
```

### Parameters

*pDecHuffState*      Pointer to the `IppiDecodeHuffmanState` structure.

### Description

The function `ippiDecodeHuffmanStateInit_JPEG` is declared in the `ippj.h` file. This function initializes the `IppiDecodeHuffmanState` structure to the initial state and must be called prior to the JPEG bit stream decoding.

### Return Values

`ippStsNoErr`      Indicates no error.

`ippStsNullPtrErr`      Indicates an error condition if one or all of the specified pointers are NULL.

---

## DecodeHuffmanStateInitAlloc\_JPEG

*Allocates memory and initializes  
IppiDecodeHuffmanState structure.*

---

### Syntax

```
IppStatus ippiDecodeHuffmanStateInitAlloc_JPEG_8u(IppiDecodeHuffmanState**  
ppDecHuffState);
```

### Parameters

*ppDecHuffState*      Pointer to the `IppiDecodeHuffmanState` structure.

### Description

The function `ippiDecodeHuffmanStateInit_JPEG` is declared in the `ippj.h` file. This function allocates memory and initializes the `IppiDecodeHuffmanState` structure to the initial state. This function must be called prior to the JPEG bit stream decoding.

### Return Values

`ippStsNoErr`      Indicates no error.

`ippStsNullPtrErr`      Indicates an error condition if one or all of the specified pointers are NULL.

---

## DecodeHuffmanStateFree\_JPEG

*Frees memory allocated by*

`ippiDecodeHuffmanStateInitAlloc_JPEG`  
*function.*

---

### Syntax

```
IppStatus ippiDecodeHuffmanStateFree_JPEG_8u(IppiDecodeHuffmanState*  
    pDecHuffState);
```

### Parameters

`pDecHuffState`            Pointer to the `IppiDecodeHuffmanState` structure.

### Description

The function `ippiDecodeHuffmanStateFree_JPEG` is declared in the `ippj.h` file. This function frees memory allocated by the `ippiDecodeHuffmanStateInitAlloc_JPEG_8u` function for the `IppiDecodeHuffmanState` structure.

### Return Values

`ippStsNoErr`            Indicates no error.

---

## DecodeHuffman8x8\_JPEG

*Performs Huffman baseline decoding of 8x8 block of  
the quantized DCT Coefficients.*

---

### Syntax

```
IppStatus ippiDecodeHuffman8x8_JPEG_1u16s_C1(const Ipp8u* pSrc,  
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, Ipp16s* pLastDC,  
    int* pMarker, const IppiDecodeHuffmanSpec* pDcTable,  
    const IppiDecodeHuffmanSpec* pAcTable, IppiDecodeHuffmanState*  
    pDecHuffState);
```

## Parameters

<i>pSrc</i>	Pointer to the source buffer with the JPEG bit stream.
<i>srcLenBytes</i>	Length in bytes of the buffer for the bit stream.
<i>pSrcCurrPos</i>	Shift in bytes of the current byte in the buffer.
<i>pDst</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pLastDC</i>	Pointer to the DC coefficient of the previous 8x8 block.
<i>pMarker</i>	Pointer to a variable that will receive JPEG marker detected during decoding.
<i>pDcTable</i>	Pointer to the Huffman table for the DC coefficients.
<i>pAcTable</i>	Pointer to the Huffman table for the AC coefficients.
<i>pDecHuffState</i>	Pointer to the <code>IppiDecodeHuffmanState</code> structure.

## Description

The function `ippiDecodeHuffman8x8_JPEG` is declared in the `ippj.h` file. This function decodes an 8x8 block of the quantized DCT coefficients using *pDcTable* and *pAcTable* tables. The decoding procedure conforms to [\[ISO10918\]](#), *Annex F.2.2, Baseline Huffman Decoding Procedures*.

If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicate an error condition if <i>srcLenBytes</i> has a zero or negative value, or <i>pSrcCurrPos</i> is out of <i>srcLenBytes</i> limit.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023..1023)$ .
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGMarkerWarn</code>	Indicates a warning if a JPEG marker is detected.

---

## DecodeHuffman8x8\_Direct\_JPEG

*Directly performs Huffman baseline decoding of an 8x8 block of the quantized DCT coefficients.*

---

### Syntax

```
IppStatus ippiDecodeHuffman8x8_Direct_JPEG_1u16s_C1(const Ipp8u* pSrc,  
    int* pSrcBitsLen, Ipp16s* pDst, Ipp16s* pLastDC, int* pMarker,  
    Ipp32u* pPrefetchedBits, int* pNumValidPrefetchedBits, const  
    IppiDecodeHuffmanSpec* pDcTable, const IppiDecodeHuffmanSpec*  
    pAcTable);
```

### Parameters

<i>pSrc</i>	Pointer to the source buffer with the JPEG bit stream.
<i>pSrcBitsLen</i>	Length in bits of the buffer for the bit stream.
<i>pDst</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pLastDC</i>	Pointer to the DC coefficient of the previous 8x8 block of the same color component.
<i>pMarker</i>	Pointer to a variable that will receive JPEG marker detected during decoding.
<i>pPrefetchedBits</i>	Pointer to the prefetch buffer which contains the previously decoded bits.
<i>pNumValidPrefetchedBits</i>	Number of valid bits in the prefetch buffer.
<i>pDcTable</i>	Pointer to the Huffman table for the DC coefficients.
<i>pAcTable</i>	Pointer to the Huffman table for the AC coefficients.

### Description

The function `ippiDecodeHuffman8x8_Direct_JPEG` is declared in the `ippj.h` file. This function directly decodes an 8x8 block of the quantized DCT coefficients using *pDcTable* and *pAcTable* tables. The function does not require any additional structure for operation. The decoding procedure conforms to [\[ISO10918\]](#), *Annex F.2.2, Baseline Huffman Decoding Procedures*.

If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

*pLastDC* should be set to 0 through the function initialization or after each restart interval.  
*pMarker* should be set to 0 through the function initialization or after the found marker has been processed. *pNumValidPrefetchedBits* should be set to 0 in the following cases: 1) through the function initialization; 2) after each restart interval; 3) after each found marker has been processed.

### Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if one or all of the specified pointers are NULL.
<i>ippStsSizeErr</i>	Indicates an error condition if <i>pSrcBitsLen</i> or <i>pNumValidPrefetchedBits</i> has negative value.
<i>ippStsJPEGDCTRangeErr</i>	Indicates an error condition if the DCT coefficient is out of the allowed range (-1023..1023).
<i>ippStsJPEGOutOfBufErr</i>	Indicates an error condition if the buffer limits are exceeded.

---

## DecodeHuffman8x8\_DCFirst\_JPEG

*Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients (first scan).*

---

### Syntax

```
IppStatus ippDecodeHuffman8x8_DCFirst_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, Ipp16s* pLastDC,
    int* pMarker, int AI, IppiDecodeHuffmanSpec* pDcTable,
    IppiDecodeHuffmanState* pDecHuffState);
```

### Parameters

<i>pSrc</i>	Pointer to the buffer with the JPEG bit stream.
<i>srcLenBytes</i>	Length in bytes of the buffer for the bit stream.
<i>pSrcCurrPos</i>	Shift in bytes of the current byte in the buffer.
<i>pDst</i>	Pointer to the 8x8 block of the quantized DCT coefficients.

<i>pLastDC</i>	Pointer to the DC coefficient of the previous 8x8 block.
<i>pMarker</i>	Pointer to a variable that will receive JPEG marker detected during decoding.
<i>Al</i>	Successive approximation bit positions low, it specifies the actual point transform.
<i>pDcTable</i>	Pointer to the Huffman table for the DC coefficients.
<i>pDecHuffState</i>	Pointer to the <code>IppiDecodeHuffmanState</code> structure.

### Description

The function `ippiDecodeHuffman8x8_DCFirst_JPEG` is declared in the `ippj.h` file. This function decodes the DC coefficient from an 8x8 block of the quantized DCT coefficients, progressive mode, the first scan, using *pDcTable* table.

The decoding procedure conforms to [\[ISO10918\]](#), *Annex G.2, Progressive Decoding of the DCT*. If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023..1023)$ .
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGMarkerWarn</code>	Indicates a warning if a JPEG marker is detected.



---

## DecodeHuffman8x8\_DCRefine\_JPEG

*Performs progressive decoding of the DC coefficient from an 8x8 block of the quantized DCT coefficients.*

---

### Syntax

```
IppStatus ippiDecodeHuffman8x8_DCRefine_JPEG_1u16s_C1(const Ipp8u* pSrc,
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int* pMarker, int Al,
    IppiDecodeHuffmanState* pDecHuffState);
```

### Parameters

<i>pSrc</i>	Pointer to the buffer with the JPEG bit stream.
<i>srcLenBytes</i>	Length in bytes of the buffer for the bit stream.
<i>pSrcCurrPos</i>	Shift in bytes of the current byte in the buffer.
<i>pDst</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pMarker</i>	Pointer to a variable that will receive a JPEG marker detected during decoding.
<i>Al</i>	Successive approximation bit positions low, it specifies the actual point transform.
<i>pDecHuffState</i>	Pointer to the <code>IppiDecodeHuffmanState</code> structure.

### Description

The function `ippiDecodeHuffman8x8_DCRefine_JPEG` is declared in the `ippj.h` file. This function decodes the DC coefficient from an 8x8 block of the quantized DCT coefficients, progressive mode, the subsequent scans, using `pDcTable` table.

The decoding procedure conforms to [\[ISO10918\]](#), Annex G.2, *Progressive Decoding of the DCT*. If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.

`ippStsJPEGDCTRangeErr` Indicates an error condition if the DCT coefficient is out of the allowed range (-1023..1023).

`ippStsJPEGOutOfBufErr` Indicates an error condition if the buffer limits are exceeded.

`ippStsJPEGMarkerWarn` Indicates a warning if a JPEG marker is detected.

---

## DecodeHuffman8x8\_ACFirst\_JPEG

*Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (first scan).*

---

### Syntax

```
IppStatus ippiDecodeHuffman8x8_ACFirst_JPEG_1u16s_C1(const Ipp8u* pSrc,  
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int* pMarker,  
    int Ss, int Se, int Al, IppiDecodeHuffmanSpec* pAcTable,  
    IppiDecodeHuffmanState* pDecHuffState);
```

### Parameters

<i>pSrc</i>	Pointer to the buffer with the JPEG bit stream.
<i>srcLenBytes</i>	Length in bytes of the buffer for the bit stream.
<i>pSrcCurrPos</i>	Shift in bytes of the current byte in the buffer.
<i>pDst</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pMarker</i>	Pointer to a variable that will receive JPEG marker detected during decoding.
<i>Ss</i>	Spectral selection start index.
<i>Se</i>	Spectral selection end index.
<i>Al</i>	Successive approximation bit positions low, it specifies the actual point transform.
<i>pAcTable</i>	Pointer to the Huffman table for the AC coefficients.
<i>pDecHuffState</i>	Pointer to the <code>IppiDecodeHuffmanState</code> structure.

## Description

The function `ippiDecodeHuffman8x8_ACFirst_JPEG` is declared in the `ippj.h` file. This function decodes AC coefficients from an 8x8 block of the quantized DCT coefficients, progressive mode, the first scan, using `pAcTable` table.

The decoding procedure conforms to [\[ISO10918\]](#), *Annex G.2, Progressive Decoding of the DCT*. If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by `pMarker`.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023 \dots 1023)$ .
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGMarkerWarn</code>	Indicates a warning if a JPEG marker is detected.

---

## DecodeHuffman8x8\_ACRefine\_JPEG

*Performs progressive decoding of the AC coefficients from an 8x8 block of the quantized DCT coefficients (subsequent scans).*

---

## Syntax

```
IppStatus ippiDecodeHuffman8x8_ACRefine_JPEG_1u16s_C1(const Ipp8u* pSrc,  
    int srcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int* pMarker,  
    int Ss, int Se, int Al, IppiDecodeHuffmanSpec* pAcTable,  
    IppiDecodeHuffmanState* pDecHuffState);
```

## Parameters

<code>pSrc</code>	Pointer to the buffer with the JPEG bit stream.
<code>srcLenBytes</code>	Length in bytes of the buffer for the bit stream.
<code>pSrcCurrPos</code>	Shift in bytes of the current byte in the buffer.

<i>pDst</i>	Pointer to the 8x8 block of the quantized DCT coefficients.
<i>pMarker</i>	Pointer to a variable that will receive JPEG marker detected during of decoding.
<i>Ss</i>	Spectral selection start index.
<i>Se</i>	Spectral selection end index.
<i>Al</i>	Successive approximation bit positions low, it specifies the actual point transform.
<i>pAcTable</i>	Pointer to the Huffman table for the AC coefficients.
<i>pDecHuffState</i>	Pointer to the <code>IppiDecodeHuffmanState</code> structure.

### Description

The function `ippiDecodeHuffman8x8_ACRrefine_JPEG` is declared in the `ippj.h` file. This function decodes AC coefficients from an 8x8 block of the quantized DCT coefficients, progressive mode, subsequent scans, using *pAcTable* table. The decoding procedure conforms to [\[ISO10918\]](#), *Annex G.2, Progressive Decoding of the DCT*.

If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one or all of the specified pointers are NULL.
<code>ippStsJPEGDCTRangeErr</code>	Indicates an error condition if the DCT coefficient is out of the allowed range $(-1023..1023)$ .
<code>ippStsJPEGOutOfBufErr</code>	Indicates an error condition if the buffer limits are exceeded.
<code>ippStsJPEGMarkerWarn</code>	Indicates a warning if a JPEG marker is detected.

## Functions for Lossless JPEG Coding

This section describes the functions that are specific for the lossless JPEG process with Huffman coding. These functions are listed in [Table 15-9](#).

**Table 15-9**      **Functions for Lossless JPEG Coding**

Function Base Name	Description
<a href="#">DiffPredFirstRow_JPEG</a>	Computes the differences between input sample and predictor for the first line
<a href="#">DiffPredRow_JPEG</a>	Computes the differences between input sample and predictor for all lines but the first.
<a href="#">ReconstructPredFirstRow_JPEG</a>	Reconstructs samples from the decoded differences between input samples and predictor for the first line.
<a href="#">ReconstructPredRow_JPEG</a>	Reconstructs samples from the decoded differences between input samples and predictor for all lines but the first.
<a href="#">GetHuffmanStatisticsOne_JPEG</a>	Computes Huffman symbol statistics
<a href="#">EncodeHuffmanOne_JPEG</a>	Performs Huffman encoding of one difference.
<a href="#">DecodeHuffmanOne_JPEG</a>	Decodes one Huffman coded difference.

The Intel IPP functions use the coding model in accordance with [[ISO10918](#)], *Annex H, Lossless Mode of Operation*.

---

### DiffPredFirstRow\_JPEG

*Computes the differences between input sample and predictor for the first line.*

---

#### Syntax

```
IppStatus ippiDiffPredFirstRow_JPEG_16s_C1(const Ipp16s* pSrc,  
      Ipp16s* pDst, int width, int P, int Pt);
```

#### Parameters

<i>pSrc</i>	Pointer to the row of samples.
<i>pDst</i>	Pointer to the row of calculated differences.
<i>width</i>	Row width in elements; has always the same value for all rows.

<i>P</i>	Sample precision derived from JPEG frame header, varies from 2 to 16.
<i>Pt</i>	Point transformation parameter derived from JPEG scan header; its value should be 0 or a positive integer.

## Description

The function `ippiDiffPredFirstRow_JPEG` is declared in the `ippj.h` file. This function operates with the first row of samples *pSrc* at the start of the scan and at the beginning of each restart interval only. It computes modulo  $2^{16}$  differences between input samples and predictor with the specified sample precision *P* and performs the point transformation with the specified parameter *Pt*. The function uses special predictors for the first line.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>width</i> parameter has a negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>P</i> or <i>Pt</i> has an illegal value.

---

## DiffPredRow\_JPEG

*Computes the differences between input sample and predictor for all lines but the first.*

---

## Syntax

```
IppStatus ippiDiffPredRow_JPEG_16s_C1(const Ipp16s* pSrc,
    const Ipp16s* pPrevRow, Ipp16s* pDst, int width, int predictor);
```

## Parameters

<i>pSrc</i>	Pointer to the row of samples.
<i>pPrevRow</i>	Pointer to the adjacent row of samples just above the current row.
<i>pDst</i>	Pointer to the row of calculated differences.
<i>width</i>	Row width in elements; has always the same value for all rows.
<i>predictor</i>	Selected predictor, should be in the range [1, 7].

## Description

The function `ippiDiffPredRow_JPEG` is declared in the `ippj.h` file. This function operates on all rows of samples except the first row at the start of the scan and at the beginning of the restart interval. It computes modulo  $2^{16}$  differences between input samples *pSrc* and the specified predictor *predictor*. The first sample of the row use the sample from the row *pPrevRow* above as a predictor.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>width</i> parameter has a negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>predictor</i> has an illegal value.

---

## ReconstructPredFirstRow\_JPEG

*Reconstructs samples from the decoded differences between input samples and predictor for the first line.*

---

## Syntax

```
IppStatus ippiReconstructPredFirstRow_JPEG_16s_C1(const Ipp16s* pSrc,
    Ipp16s* pDst, int width, int P, int Pt);
```

## Parameters

<i>pSrc</i>	Pointer to the row of decoded differences.
<i>pDst</i>	Pointer to the row of reconstructed samples.
<i>width</i>	Row width in elements; has always the same value for all rows.
<i>P</i>	Sample precision derived from JPEG frame header, varies from 2 to 16.
<i>Pt</i>	Point transformation parameter derived from JPEG scan header; its value should be 0 or a positive integer.

## Description

The function `ippiReconstructPredFirstRow_JPEG` is declared in the `ippj.h` file. This function operates on the first row of decoded differences *pSrc* at the start of the scan and restart interval only. It reconstructs output samples *pDst* with the specified sample precision *P* adding decoded differences modulo  $2^{16}$  to the predictions. It also performs the point transformation with the specified parameter *Pt*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>width</i> parameter has a negative value.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>P</i> or <i>Pt</i> has an illegal value.

---

## ReconstructPredRow\_JPEG

*Reconstructs samples from the decoded differences between input samples and predictor for all lines but the first.*

---

## Syntax

```
IppStatus ippiReconstructPredRow_JPEG_16s_C1(const Ipp16s* pSrc, const
      Ipp16s* pPrevRow, Ipp16s* pDst, int width, int predictor);
```

## Parameters

<i>pSrc</i>	Pointer to the row of decoded differences.
<i>pPrevRow</i>	Pointer to the adjacent row of reconstructed samples just above the current row.
<i>pDst</i>	Pointer to the row of reconstructed samples.
<i>width</i>	Row width in elements (has always the same value for all rows).
<i>predictor</i>	Selected predictor, should be in the range [1, 7].



### Description

The function `ippiReconstructPredRow_JPEG` is declared in the `ippj.h` file. This function operates on all rows of decoded differences *pSrc* except first rows at the start of the scan and restart interval. It reconstructs output samples *pDst* adding decoded differences modulo  $2^{16}$  to the predictions specified by the *predictor* parameters. The first sample of the row use the reconstructed sample from the row *pPrevRow* above as a predictor.

### Return Values

<code>ippiStsNoErr</code>	Indicates no error.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippiStsSizeErr</code>	Indicates an error condition if the <i>width</i> parameter has a negative value.
<code>ippiStsBadArgErr</code>	Indicates an error condition if <i>predictor</i> has an illegal value.

---

## GetHuffmanStatisticsOne\_JPEG

*Computes Huffman symbol statistics.*

---

### Syntax

```
IppStatus ippiGetHuffmanStatisticsOne_JPEG_16s_C1(const Ipp16s* pSrc,  
int pHuffStatistics[256]);
```

### Parameters

<i>pSrc</i>	Pointer to the row of decoded differences.
<i>pHuffStatistics</i>	Pointer to the calculated statistics buffer.

### Description

The function `ippiGetHuffmanStatisticsOne_JPEG` is declared in the `ippj.h` file. This function computes frequency of Huffman symbols to be encoded. Based on these statistics, an optimal Huffman table can be built by the `ippiEncodeHuffmanRwaTableInit_JPEG` function. This table saves space by removing unused symbols and assigning shorter codes for more frequent symbols.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

---

## EncodeHuffmanOne\_JPEG

*Performs Huffman encoding of one difference.*

---

### Syntax

```
IppStatus ippiEncodeHuffmanOne_JPEG_16s1u_C1(const Ipp16s* pSrc,  
        Ipp8u* pDst, int nDstLenBytes, int* pDstCurrPos,  
        const IppiEncodeHuffmanSpec* pHuffTable,  
        IppiEncodeHuffmanState* pEncHuffState, int bFlushState);
```

### Parameters

<i>pSrc</i>	Pointer to the difference to be encoded. This pointer can be NULL if <i>bFlushState</i> = 1.
<i>pDst</i>	Pointer to the output bitstream buffer.
<i>nDstLenBytes</i>	Number of available bytes in the output buffer.
<i>pDstCurrPos</i>	Pointer to the current byte in the output buffer. This pointer is updated in the function.
<i>pHuffTable</i>	Pointer to the <code>IppiEncodeHuffmanSpec</code> structure that contains the Huffman code table. This pointer can be NULL if <i>bFlushState</i> = 1.
<i>pEncHuffState</i>	Pointer to the <code>IppiEncodeHuffmanState</code> structure that contains the Huffman encoder state.
<i>bFlushState</i>	Setting this parameter to 1 forces the function to flush collected bits from the state structure to the bitstream, setting it to 0 forces to perform encoding.

### Description

The function `ippiEncodeHuffmanOne_JPEG` is declared in the `ippj.h` file. This function encodes one difference *pSrc* using Huffman code table *pHuffTable*. Only full bytes are written to the output buffer. The `IppiEncodeHuffmanState` structure collects the bits that do not make

up a complete byte. To force adding the bits accumulated in the `IppiEncodeHuffmanState` to the output buffer, the parameter `bFlushState` should be set to 1 when the last sample in scan or restart interval is encoded. In all other cases this parameter must be set to zero.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	When <code>bFlushState=0</code> - Indicates an error condition if one of the specified pointers is NULL, when <code>bFlushState=1</code> - Indicates an error condition if one of the <code>pDst</code> , <code>pDstCurrPos</code> or <code>pEncHuffState</code> pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the <code>nDstLenBytes</code> parameter has zero or negative value, or if <code>pDstCurrPos</code> is out of <code>nDstLenBytes</code> limit.

---

## DecodeHuffmanOne\_JPEG

*Decodes one Huffman coded difference.*

---

### Syntax

```
IppStatus ippiDecodeHuffmanOne_JPEG_1u16s_C1(const Ipp8u* pSrc,
        int nSrcLenBytes, int* pSrcCurrPos, Ipp16s* pDst, int* pMarker,
        const IppiDecodeHuffmanSpec* pHuffTable,
        IppiDecodeHuffmanState* pDecHuffState);
```

### Parameters

<code>pSrc</code>	Pointer to the input bitstream.
<code>nSrcLenBytes</code>	Number of available bytes in the input buffer.
<code>pSrcCurrPos</code>	Pointer to the current byte in the input buffer. This pointer is updated in the function.
<code>pDst</code>	Pointer to the output buffer to store the decoded difference.
<code>pMarker</code>	Pointer to a variable that will receive JPEG marker detected during decoding.
<code>pHuffTable</code>	Pointer to the <code>IppiDecodeHuffmanSpec</code> structure that contains the Huffman code table.

*pDecHuffState* Pointer to the `IppiDecodeHuffmanState` structure that contains the Huffman decoder state.

## Description

The function `ippiDecodeHuffmanOne_JPEG` is declared in the `ippj.h` file. This function decodes one Huffman coded difference pointed to in the bitstream by *pSrc* and stores the result in the output buffer *pDst*. The function uses Huffman code table *pHuffTable*.

If a JPEG marker is detected during decoding, the function stops decoding and writes the marker to a location indicated by *pMarker*.

## Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error condition if one of the specified pointers is `NULL`.

`ippStsSizeErr` Indicates an error condition if the *nSrcLenBytes* parameter has zero or negative value, or if *pSrcCurrPos* is out of *nSrcLenBytes* limit.

`ippStsJPEGMarkerWarn` Indicates a warning if a JPEG marker is detected.

# Wavelet Transform Functions

This section describes the wavelet transform functions that are specific for JPEG 2000 image coding system (see [ISO15444]). These functions are listed in the [Table 15-10](#).

**Table 15-10 Wavelet Transform Functions**

Function Base Name	Description
<b>Low-Level Operations</b>	
<a href="#">WTFwdRow_B53_JPEG2K</a>	Performs a row oriented forward wavelet transform with reversible filter.
<a href="#">WTInvRow_B53_JPEG2K</a>	Performs a row oriented inverse wavelet transform with reversible filter.
<a href="#">WTFwdCol_B53_JPEG2K</a>	Performs a forward wavelet transform with reversible filter of image columns.
<a href="#">WTFwdColLift_B53_JPEG2K</a>	Performs a single step of forward wavelet transform with reversible filter of image columns.
<a href="#">WTInvCol_B53_JPEG2K</a>	Performs a column oriented inverse wavelet transform with reversible filter.

**Table 15-10 Wavelet Transform Functions** (continued)

Function Base Name	Description
<a href="#"><u>WTInvColLift_B53_JPEG2K</u></a>	Performs a single step of inverse wavelet transform with reversible filter of image columns.
<a href="#"><u>WTFwdRow_D97_JPEG2K</u></a>	Performs a row oriented forward wavelet transform with irreversible filter.
<a href="#"><u>WTInvRow_D97_JPEG2K</u></a>	Performs a row oriented inverse wavelet transform with irreversible filter.
<a href="#"><u>WTFwdCol_D97_JPEG2K</u></a>	Performs a column oriented forward wavelet transform with irreversible filter.
<a href="#"><u>WTFwdColLift_D97_JPEG2K</u></a>	Performs a single step of forward wavelet transform with irreversible filter of image columns.
<a href="#"><u>WTInvCol_D97_JPEG2K</u></a>	Performs a column oriented inverse wavelet transform with irreversible filter.
<a href="#"><u>WTInvColLift_D97_JPEG2K</u></a>	Performs a single step of inverse wavelet transform with irreversible filter of image columns.
<b>Tile-Oriented Transforms</b>	
<a href="#"><u>WTGetBufSize_B53_JPEG2K</u></a>	Computes the size of the buffer for a tile-oriented wavelet transform
<a href="#"><u>WTFwd_B53_JPEG2K</u></a>	Performs a tile-oriented forward wavelet transform.
<a href="#"><u>WTInv_B53_JPEG2K</u></a>	Performs a tile-oriented inverse wavelet transform.
<a href="#"><u>WTGetBufSize_D97_JPEG2K</u></a>	Computes the size of the buffer for a tile-oriented wavelet transform
<a href="#"><u>WTFwd_D97_JPEG2K</u></a>	Performs a tile-oriented forward wavelet transform.
<a href="#"><u>WTInv_D97_JPEG2K</u></a>	Performs a tile-oriented inverse wavelet transform.

## Low-Level Operations

These functions perform one-dimensional reversible (`_B53` modifier) and irreversible (`_D97` modifier) wavelet transforms of image rows or columns for lossless and lossy compressions, respectively, in accordance with [ISO15444], *Annex F, Discrete Wavelet Transformation of Tile-Components*. The transformations are the lifting-based implementations of filtering by 5-3 reversible and 9-7 irreversible wavelet filters.

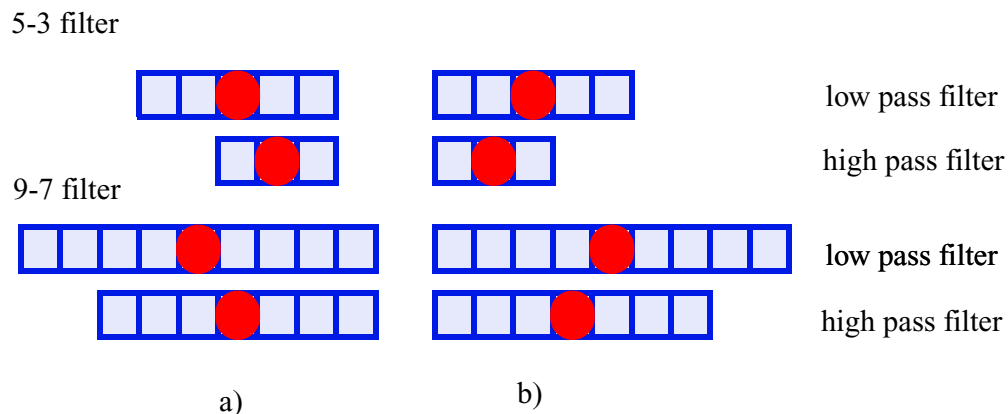
The filters have odd length: 5 (9) - for lowpass filters, and 3 (7) - for highpass filters. There are two possible relative positions of lowpass and highpass filters allowed by the standard, as shown in the [Figure 15-2](#). The relative position is specified by the *phase* argument that has two possible values:

`ippWTFilterFirstLow` corresponds to positions a),  
`ippWTFilterFirstHigh` corresponds to positions b).

The wavelet transform functions do not apply any fixed border extension (symmetrical, wraparound or other) to the source image ROI, instead they require valid and accessible border data outside of the source image ROI.

**Figure 15-2 Possible Relative Positions of Filters**

---



## WTFwdRow\_B53\_JPEG2K

*Performs a forward wavelet transform with reversible filter of image rows.*

### Syntax

```
IppStatus ippiWTFwdRow_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc,
    int srcStep, Ipp16s* pDstLow, int dstLowStep, Ipp16s* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);

IppStatus ippiWTFwdRow_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc,
    int srcStep, Ipp32s* pDstLow, int dstLowStep, Ipp32s* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstLow</i>	Pointer to ROI of the low frequency component of a destination image.
<i>dstLowStep</i>	Distance in bytes between starts of consecutive lines in the low frequency component of a destination image.
<i>pDstHigh</i>	Pointer to ROI of the high frequency component of a destination image.
<i>dstHighStep</i>	Distance in bytes between starts of consecutive lines in the high frequency component of a destination image.
<i>dstRoiSize</i>	Size of the destination image ROI.
<i>phase</i>	Relative position of the high-pass and low-pass filters (see <a href="#">page 15-117</a> ).

### Description

The function `ippiWTFwdRow_B53_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet decomposition of the source image rows using the 5-3 reversible filter. Both destination ROIs have the same size `dstRoiSize`, while the source image ROI size is uniquely determined from the following relations:

```
srcRoiSize.width = 2 * dstRoiSize.width;
```

```
srcRoiSize.height = dstRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 2 extra pixels on the left and 1 pixel on the right (outside ROI border) for each processed row;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 1 extra pixel on the left and 2 pixels on the right (outside ROI border) for each processed row.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

---

## WTInvRow\_B53\_JPEG2K

*Performs an inverse wavelet transform with reversible filter of image rows.*

---

### Syntax

```
IppStatus ippiWTInvRow_B53_JPEG2K_16s_C1R(const Ipp16s* pSrcLow,
    int srcLowStep, const Ipp16s* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp16s* pDst, int dstStep, IppiWTFilterFirst phase);

IppStatus ippiWTInvRow_B53_JPEG2K_32s_C1R(const Ipp32s* pSrcLow,
    int srcLowStep, const Ipp32s* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiWTFilterFirst phase);
```

### Parameters

<i>pSrcLow</i>	Pointer to ROI of the low frequency component of a source image.
<i>srcLowStep</i>	Distance in bytes between starts of consecutive lines in the low frequency component of a source image.



<i>pSrcHigh</i>	Pointer to ROI of the high frequency component of a source image.
<i>srcHighStep</i>	Distance in bytes between starts of consecutive lines in the high frequency component of a source image.
<i>srcRoiSize</i>	Size of the source image ROI.
<i>pDst</i>	Pointer to ROI of the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>phase</i>	Relative position of the high-pass and low-pass filters (see <a href="#">page 15-117</a> ).

## Description

The function `ippiWTInvRow_B53_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet reconstruction of the source image rows using the 5-3 reversible filter. Both source ROIs have the same size *srcRoiSize*, while the destination image ROI size is uniquely determined from the following relations:

```
dstRoiSize.width = 2 * srcRoiSize.width;
dstRoiSize.height = srcRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI.

For the low frequency component:

if the *phase* argument is equal to `ippWTFILTERFirstLow`, the function requires 1 extra pixel on the right (outside ROI border) for each processed row;

if the *phase* argument is equal to `ippWTFILTERFirstHigh`, the function requires 1 extra pixel on the left (outside ROI border) for each processed row.

For the high frequency component, the function always requires 1 extra pixel on the left and 1 pixel on the right (outside ROI border) for each processed row.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.

<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.
----------------------------	--

---

## WTFwdCol\_B53\_JPEG2K

*Performs a forward wavelet transform with reversible filter of image columns.*

---

### Syntax

```
IppStatus ippiWTFwdCol_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc,  
    int srcStep, Ipp16s* pDstLow, int dstLowStep, Ipp16s* pDstHigh,  
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);  
IppStatus ippiWTFwdCol_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc,  
    int srcStep, Ipp32s* pDstLow, int dstLowStep, Ipp32s* pDstHigh,  
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstLow</code>	Pointer to ROI of the low frequency component of a destination image.
<code>dstLowStep</code>	Distance in bytes between starts of consecutive lines in the low frequency component of a destination image.
<code>pDstHigh</code>	Pointer to ROI of the high frequency component of a destination image.
<code>dstHighStep</code>	Distance in bytes between starts of consecutive lines in the high frequency component of a destination image.
<code>dstRoiSize</code>	Size of the destination image ROI.
<code>phase</code>	Relative position of the high-pass and low-pass filters (see <a href="#">page 15-116</a> ).

### Description

The function `ippiWTFwdCol_B53_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet decomposition of the source image columns using the 5-3 reversible filter. Both destination ROIs have the same size *dstRoiSize*, while the source image ROI size is uniquely determined from the following relations:

```
srcRoiSize.width = dstRoiSize.width;
srcRoiSize.height = 2 * dstRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 2 extra pixels up and 1 pixel down (outside ROI border) for each processed column;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 1 extra pixel up and 2 pixels down (outside ROI border) for each processed column.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

---

## WTFwdColLift\_B53\_JPEG2K

*Performs a single step of forward wavelet transform with reversible filter of image columns.*

---

### Syntax

```
IppStatus ippWTFwdColStepLift_B53_JPEG2K_16s_C1(const Ipp16s* pSrc0,
    const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDstLow0, const
    Ipp16s* pSrcHigh0, Ipp16s* pDstHigh1, int width);

IppStatus ippWTFwdColStepLift_B53_JPEG2K_32s_C1(const Ipp32s* pSrc0,
    const Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s* pDstLow0, const
    Ipp32s* pSrcHigh0, Ipp32s* pDstHigh1, int width);
```

## Parameters

<i>pSrc0</i>	Pointer to the source image row #0.
<i>pSrc1</i>	Pointer to the source image row #1.
<i>pSrc2</i>	Pointer to the source image row #2.
<i>pSrcHigh0</i>	Pointer to the high frequency component row #0 of a source image.
<i>pDstLow0</i>	Pointer to the low frequency component row #0 of a destination image.
<i>pDstHigh1</i>	Pointer to the high frequency component row #1 of a destination image.
<i>width</i>	Width in pixels of rows.

## Description

The function `ippiWTFwdColLift_B53_JPEG2K` is declared in the `ippj.h` file. This function performs a single step of row-scan-based wavelet decomposition of the source image columns using the 5-3 reversible filter.

For each *i*-pixel in the row this function computes the following values:

$$pDstHigh1[i] = pSrc1[i] - \left\lfloor \frac{pSrc0[i] + pSrc2[i]}{2} \right\rfloor$$

$$pDstLow0[i] = pSrc1[i] + \left\lfloor \frac{pSrcHigh0[i] + pDstHigh1[i] + 2}{4} \right\rfloor$$

Here the notation  $\lfloor x \rfloor$  designates the floor function, that is, the largest integer not exceeding *x*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width of row has zero or negative value.

---

## WTInvCol\_B53\_JPEG2K

*Performs an inverse wavelet transform with reversible filter of image columns.*

---

### Syntax

```
IppStatus ippiWTInvCol_B53_JPEG2K_16s_C1R(const Ipp16s* pSrcLow,
    int srcLowStep, const Ipp16s* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp16s* pDst, int dstStep, IppiWTFilterFirst phase);
IppStatus ippiWTInvCol_B53_JPEG2K_32s_C1R(const Ipp32s* pSrcLow,
    int srcLowStep, const Ipp32s* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiWTFilterFirst phase);
```

### Parameters

<i>pSrcLow</i>	Pointer to ROI of the low frequency component of a source image.
<i>srcLowStep</i>	Distance in bytes between starts of consecutive lines in the low frequency component of a source image.
<i>pSrcHigh</i>	Pointer to ROI of the high frequency component of a source image.
<i>srcHighStep</i>	Distance in bytes between starts of consecutive lines in the high frequency component of a source image.
<i>srcRoiSize</i>	Size of the source image ROI.
<i>pDst</i>	Pointer to ROI of the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>phase</i>	Relative position of the high-pass and low-pass filters (see <a href="#">page 15-116</a> ).

### Description

The function `ippiWTInvCol_B53_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet reconstruction of the source image columns using the 5-3 reversible filter. Both source image ROIs have the same size *srcRoiSize*, while the destination image ROI size is uniquely determined from the following relations:

```
dstRoiSize.width = srcRoiSize.width;
```

```
dstRoiSize.height = 2 * srcRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI.

For the low frequency component:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 1 extra pixel down (outside ROI border) for each processed column;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 1 extra pixel up (outside ROI border) for each processed column.

For the high frequency component, the function always requires 1 extra pixel up and 1 pixel down (outside ROI border) for each processed column.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

---

## WTInvColLift\_B53\_JPEG2K

*Performs a single step of inverse wavelet transform with reversible filter of image columns.*

---

### Syntax

```
IppStatus ippiWTInvColStepLift_B53_JPEG2K_16s_C1(const Ipp16s* pSrcLow0,
    const Ipp16s* pSrcHigh0, const Ipp16s* pSrcHigh1, const Ipp16s*
    pSrc0, Ipp16s* pDst1, Ipp16s* pDst2, int width);

IppStatus ippiWTInvColStepLift_B53_JPEG2K_32s_C1(const Ipp32s* pSrcLow0,
    const Ipp32s* pSrcHigh0, const Ipp32s* pSrcHigh1, const Ipp32s*
    pSrc0, Ipp32s* pDst1, Ipp32s* pDst2, int width)
```

### Parameters

*pSrcLow0*                      Pointer to the source low frequency component row #0.

<i>pSrcHigh0</i>	Pointer to the source high frequency component row #0.
<i>pSrcHigh1</i>	Pointer to the source high frequency component row #1.
<i>pSrc0</i>	Pointer to the reconstructed image row #0.
<i>pDst1</i>	Pointer to the reconstructed image row #1.
<i>pDst2</i>	Pointer to the reconstructed image row #2.
<i>width</i>	Width in pixels of rows.

## Description

The function `ippiWTInvColLift_B53_JPEG2K` is declared in the `ippj.h` file. This function performs a single step of row-scan-based wavelet reconstruction of the source image columns using the 5-3 reversible filter.

For each  $i$ -pixel in the row this function computes the following values:

$$pDst2[i] = pSrcLow0[i] - \left\lfloor \frac{pSrcHigh0[i] + pSrcHigh1[i] + 2}{4} \right\rfloor$$

$$pDst1[i] = pSrcHigh0[i] + \left\lfloor \frac{pSrc0[i] + pDst2[i]}{2} \right\rfloor$$

Here the notation  $\lfloor x \rfloor$  designates the floor function, that is, the largest integer not exceeding  $x$ .

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the <i>width</i> is less than or equal to 0.

---

## WTFwdRow\_D97\_JPEG2K

*Performs a forward wavelet transform with irreversible filter of image rows.*

---

### Syntax

```
IppStatus ippiWTFwdRow_D97_JPEG2K_<mod>(const Ipp<datatype>* pSrc,  
    int srcStep, Ipp<datatype>* pDstLow, int dstLowStep, Ipp<datatype>*  
    pDstHigh, int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst  
    phase);
```

Supported values for *mod*:

16s\_C1R      32s\_C1R      32f\_C1R

### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDstLow</i>	Pointer to ROI of the low frequency component of a destination image.
<i>dstLowStep</i>	Distance in bytes between starts of consecutive lines in the low frequency component of a destination image.
<i>pDstHigh</i>	Pointer to ROI of the high frequency component of a destination image.
<i>dstHighStep</i>	Distance in bytes between starts of consecutive lines in the high frequency component of a destination image.
<i>dstRoiSize</i>	Size of the destination image ROI.
<i>phase</i>	Relative position of the high-pass and low-pass filters (see <a href="#">page 15-116</a> ).

### Description

The function `ippiWTFwdRow_D97_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet decomposition of the source image rows using the 9-7 irreversible filter. Both destination ROIs have the same size *dstRoiSize*, while the source image ROI size is uniquely determined from the following relations:



```
srcRoiSize.width = 2 * dstRoiSize.width;
srcRoiSize.height = dstRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI:

if the *phase* argument is equal to `ippWTFILTERFIRSTLOW`, the function requires 4 extra pixels on the left and 3 pixels on the right (outside ROI border) for each processed row;

if the *phase* argument is equal to `ippWTFILTERFIRSHIGH`, the function requires 3 extra pixels on the left and 4 pixels on the right (outside ROI border) for each processed row.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

---

## WTInvRow\_D97\_JPEG2K

*Performs an inverse wavelet transform with irreversible filter of image rows.*

---

### Syntax

```
IppStatus ippWTInvRow_D97_JPEG2K<mod>(const Ipp<datatype>* pSrcLow,
    int srcLowStep, const Ipp<datatype>* pSrcHigh, int srcHighStep,
    IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiWTFILTERFIRST
    phase);
```

Supported values for *mod*:

```
16s_C1R    32s_C1R    32f_C1R
```

### Parameters

*pSrcLow*                      Pointer to ROI of the low frequency component of a source image.

<i>srcLowStep</i>	Distance in bytes between starts of consecutive lines in the low frequency component of a source image.
<i>pSrcHigh</i>	Pointer to ROI of the high frequency component of a source image.
<i>srcHighStep</i>	Distance in bytes between starts of consecutive lines in the high frequency component of a source image.
<i>srcRoiSize</i>	Size of the source image ROI.
<i>pDst</i>	Pointer to ROI of the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>phase</i>	Relative position of the high-pass and low-pass filters (see <a href="#">page 15-116</a> ).

### Description

The function `ippiWTInvRow_D97_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet reconstruction of the source image rows using the 9-7 irreversible filter. Both source ROIs have the same size *srcRoiSize*, while the destination image ROI size is uniquely determined from the following relations:

```
dstRoiSize.width = 2 * srcRoiSize.width;  
dstRoiSize.height = srcRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI.

For the low frequency component:

- if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 1 extra pixel on the left and 2 extra pixels on the right (outside ROI border) for each processed row;
- if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 2 extra pixels on the left and 1 extra pixel on the right (outside ROI border) for each processed row.

For the high frequency component, the function always requires 2 extra pixels on the left and 2 extra pixels on the right (outside ROI border) for each processed row.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

---

## WTFwdCol\_D97\_JPEG2K

*Performs a forward wavelet transform with irreversible filter of image columns.*

---

### Syntax

```
IppStatus ippWTFwdCol_D97_JPEG2K_32f_C1R(const Ipp32f* pSrc,
    int srcStep, Ipp32f* pDstLow, int dstLowStep, Ipp32f* pDstHigh,
    int dstHighStep, IppiSize dstRoiSize, IppiWTFilterFirst phase);
```

### Parameters

<code>pSrc</code>	Pointer to the source image ROI.
<code>srcStep</code>	Distance in bytes between starts of consecutive lines in the source image.
<code>pDstLow</code>	Pointer to ROI of the low frequency component of a destination image.
<code>dstLowStep</code>	Distance in bytes between starts of consecutive lines in the low frequency component of a destination image.
<code>pDstHigh</code>	Pointer to ROI of the high frequency component of a destination image.
<code>dstHighStep</code>	Distance in bytes between starts of consecutive lines in the high frequency component of a destination image.
<code>dstRoiSize</code>	Size of the destination image ROI.
<code>phase</code>	Relative position of the high-pass and low-pass filters (see <a href="#">page 15-116</a> ).

### Description

The function `ippWTFwdCol_D97_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet decomposition of the source image columns using the 9-7 irreversible filter. Both destination ROIs have the same size *dstRoiSize*, while the source image ROI size is uniquely determined from the following relations:

```
srcRoiSize.width = dstRoiSize.width;
srcRoiSize.height = 2 * dstRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 4 extra pixels up and 3 extra pixels down (outside ROI border) for each processed column;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 3 extra pixels up and 4 extra pixels down (outside ROI border) for each processed column.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

---

## WTFwdColLift\_D97\_JPEG2K

*Performs a single step of forward wavelet transform with irreversible filter of image columns.*

---

### Syntax

```
IppStatus ippiWTFwdColLift_D97_JPEG2K_<mod>(const Ipp<datatype>* pSrc0,
const Ipp<datatype>* pSrc1, const Ipp<datatype>* pSrc2,
Ipp<datatype>* pSrcDstLow0, Ipp<datatype>* pDstLow1, Ipp<datatype>*
pSrcDstHigh0, Ipp<datatype>* pSrcDstHigh1, Ipp<datatype>* pDstHigh2,
int width);
```

Supported values for *mod*:

```
16s_C1      32s_C1      32f_C1
```

## Parameters

<code>pSrc0</code>	Pointer to the source image row #0.
<code>pSrc1</code>	Pointer to the source image row #1.
<code>pSrc2</code>	Pointer to the source image row #2.
<code>pSrcDstLow0</code>	Pointer to the low frequency component row #0.
<code>pDstLow1</code>	Pointer to the low frequency component row #1.
<code>pSrcDstHigh0</code>	Pointer to the high frequency component row #0.
<code>pSrcDstHigh1</code>	Pointer to the high frequency component row #1.
<code>pDstHigh2</code>	Pointer to the high frequency component row #2.
<code>width</code>	Width in pixels of rows.

## Description

The function `ippiWTFwdColLift_D97_JPEG2K` is declared in the `ippj.h` file. This function performs a single step of row-scan-based 2D wavelet decomposition of the source image columns using the 9-7 irreversible filter.

For each  $i$ -pixel in the row this function computes the following values:

$$\begin{aligned}
 pDstHigh2[i] &= pSrc1[i] + \alpha \cdot (pSrc0[i] + pSrc2[i]) \\
 pDstLow1[i] &= pSrc0[i] + \beta \cdot (pSrcDstHigh1[i] + pDstHigh2[i]) \\
 pSrcDstHigh1[i] &= pSrcDstHigh1[i-1] + \gamma \cdot (pSrcDstLow0[i] + pDstLow1[i]) \\
 pSrcDstLow0[i] &= pSrcDstLow0[i-1] + \delta \cdot (pSrcDstHigh0[i] + pSrcDstHigh1[i]) \\
 pSrcDstLow0[i] &= pSrcDstLow0[i] \cdot K \\
 pSrcDstHigh0[i] &= (pSrcDstHigh0[i]) / K
 \end{aligned}$$

where  $\alpha, \beta, \gamma, \delta, K$  - are the lifting parameters in accordance with [\[ISO15444\]](#), Annex F.3, *Inverse Discrete Wavelet Transformation*.

## Return Values

<code>ippiStsNoErr</code>	Indicates no error.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippiStsSizeErr</code>	Indicates an error condition if the <code>width</code> is less than or equal to 0.

---

## WTInvCol\_D97\_JPEG2K

*Performs an inverse wavelet transform with irreversible filter of image columns.*

---

### Syntax

```
IppStatus ippiWTInvCol_D97_JPEG2K_32f_C1R(const Ipp32f* pSrcLow,
      int srcLowStep, const Ipp32f* pSrcHigh, int srcHighStep,
      IppiSize srcRoiSize, Ipp32f* pDst, int dstStep,
      IppiWTFilterFirst phase);
```

### Parameters

<i>pSrcLow</i>	Pointer to ROI of the low frequency component of a source image.
<i>srcLowStep</i>	Distance in bytes between starts of consecutive lines in the low frequency component of a source image.
<i>pSrcHigh</i>	Pointer to ROI of the high frequency component of a source image.
<i>srcHighStep</i>	Distance in bytes between starts of consecutive lines in the high frequency component of a source image.
<i>srcRoiSize</i>	Size of the source image ROI.
<i>pDst</i>	Pointer to ROI of the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>phase</i>	Relative position of the high-pass and low-pass filters (see <a href="#">page 15-116</a> ).

### Description

The function `ippiWTInvCol_D97_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs a wavelet reconstruction of the source image columns using the 9-7 irreversible filter. Both source image ROIs have the same size *srcRoiSize*, while the destination image ROI size is uniquely determined from the following relations:

```
dstRoiSize.width = srcRoiSize.width;
dstRoiSize.height = 2 * srcRoiSize.height
```

For proper operation, the function needs valid data outside the source image ROI.

For the low frequency component:

if the *phase* argument is equal to `ippWTFilterFirstLow`, the function requires 1 extra pixel up and 2 extra pixels down (outside ROI border) for each processed column;

if the *phase* argument is equal to `ippWTFilterFirstHigh`, the function requires 2 extra pixels up and 1 extra pixel down (outside ROI border) for each processed column.

For the high frequency component, the function always requires 2 extra pixels up and 2 extra pixels down (outside ROI border) for each processed column.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

---

## WTInvColLift\_D97\_JPEG2K

*Performs a single step of inverse wavelet transform with irreversible filter of image columns.*

---

### Syntax

```
IppStatus ippWTInvColLift_D97_JPEG2K_<mod>(const Ipp<datatype>*
    pSrcLow0, const Ipp<datatype>* pSrcHigh0, const Ipp<datatype>*
    pSrcHigh1, const Ipp<datatype>* pSrc0, Ipp<datatype>* pSrcDst1,
    Ipp<datatype>* pSrcDst2, Ipp<datatype>* pDst3, Ipp<datatype>* pDst4,
    int width);
```

Supported values for *mod*:

`16s_C1`      `32s_C1`      `32f_C1`

### Parameters

*pSrcLow0*      Pointer to the source low frequency component row #0.

<i>pSrcHigh0</i>	Pointer to the source high frequency component row #0.
<i>pSrcHigh1</i>	Pointer to the source high frequency component row #1.
<i>pSrc0</i>	Pointer to the reconstructed image row #0.
<i>pSrcDst1</i>	Pointer to the reconstructed image row #1.
<i>pSrcDst2</i>	Pointer to the reconstructed image row #2.
<i>pDst3</i>	Pointer to the reconstructed image row #3.
<i>pDst4</i>	Pointer to the reconstructed image row #4.
<i>width</i>	Width in pixels of rows.

### Description

The function `ippiWTInvColLift_D97_JPEG2K` is declared in the `ippj.h` file. This function performs a single step of row-scan-based 2D wavelet reconstruction of the source image columns using the 9-7 irreversible filter.

For each *i*-pixel in the row this function computes the following values:

$$\begin{aligned}
 pDst4[i] &= pSrcLow0[i] \cdot K - (\delta / K \cdot (pSrcHigh0[i] + pSrcHigh1[i])) \\
 pDst3[i] &= (pSrcHigh0[i]) / K - (\gamma \cdot (pSrcDst2[i] + pDst4[i])) \\
 pSrcDst2[i] &= pSrcDst2[i-1] - (\beta \cdot (pSrcDst1[i] + pDst3[i])) \\
 pSrcDst1[i] &= pSrcDst1[i-1] - (\alpha \cdot (pSrc0[i] + pSrcDst2[i]))
 \end{aligned}$$

where  $\alpha, \beta, \gamma, \delta, K$  - are the lifting parameters in accordance with [\[ISO15444\]](#), *Annex F.3, Inverse Discrete Wavelet Transformation*.

### Return Values

<code>ippiStsNoErr</code>	Indicates no error.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippiStsSizeErr</code>	Indicates an error condition if the <i>width</i> is less than or equal to 0.



## Tile-Oriented Transforms

These functions perform high-level tile-oriented reversible (`_B53` modifier) and irreversible (`_D97` modifier) wavelet transforms for lossless and lossy compressions, respectively, in accordance with [ISO15444], *Annex F, Discrete Wavelet Transformation of Tile-Components*. The transformations are the lifting-based implementations of filtering by 5-3 reversible and 9-7 irreversible wavelet filters.

The functions operate with tile-component and perform one-level decomposition or reconstruction of the image. Each tile-component is transformed into a set of four two-dimensional subbands.

---

## WTGetBufSize\_B53\_JPEG2K

*Computes the size of the buffer for wavelet transform with reversible filter.*

---

### Syntax

```
IppStatus ippWTGetBufSize_B53_JPEG2K_16s_C1R(const IppiRect* pTileRect,
        int* pSize);
IppStatus ippWTGetBufSize_B53_JPEG2K_32s_C1R(const IppiRect* pTileRect,
        int* pSize);
IppStatus ippWTGetBufSize_B53_JPEG2K_16s_C1IR(const IppiRect* pTileRect,
        int* pSize);
IppStatus ippWTGetBufSize_B53_JPEG2K_32s_C1IR(const IppiRect* pTileRect,
        int* pSize);
```

### Parameters

<i>pTileRect</i>	Pointer to the tile rectangle structure.
<i>pSize</i>	Pointer to the computed value of the buffer.

### Description

The function `ippWTGetBufSize_B53_JPEG2K` is declared in the `ippj.h` file. This function computes the size in bytes of the work buffer that is required for a tile-oriented wavelet transform functions [ippWTFwd\\_B53\\_JPEG2K](#) and [ippWTInv\\_B53\\_JPEG2K](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.

---

## WTFwd\_B53\_JPEG2K

*Performs a tile-oriented forward wavelet transform with reversible filter.*

---

### Syntax

```

IppStatus ippiWTFwd_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc, int srcStep,
    const IppiRect* pTileRect, Ipp16s* pDst[4], int dstStep[4], Ipp8u* pBuffer);
IppStatus ippiWTFwd_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc, int srcStep,
    const IppiRect* pTileRect, Ipp32s* pDst[4], int dstStep[4], Ipp8u* pBuffer);
IppStatus ippiWTFwd_B53_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile, int srcDstStep,
    const IppiRect* pTileRect, Ipp8u* pBuffer);
IppStatus ippiWTFwd_B53_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile, int srcDstStep,
    const IppiRect* pTileRect, Ipp8u* pBuffer);

```

### Parameters

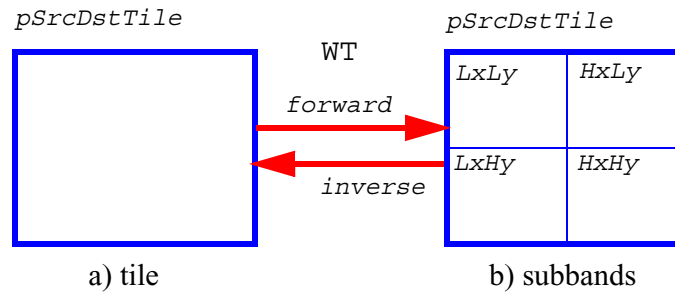
<i>pSrc</i>	Pointer to the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pTileRect</i>	Pointer to the tile rectangle structure.
<i>pDst</i>	Array of pointers to the subbands of the destination image (in the order LxLy, LxHy, HxLy, HxHy).
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the subbands of the destination image.
<i>pSrcDstTile</i>	Pointer to the source and destination image for in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

The function `ippiWTFwd_B53_JPEG2K` is declared in the `ippj.h` file. This function performs a forward discrete wavelet transformation of the tile-component using a subband decomposition in accordance with [ISO15444], *Annex F, Discrete Wavelet Transformation of Tile-Components*. The function performs one-level decomposition by downsampling each tile of the source image `pSrc` (`pSrcDstTile` for in-place flavors) into a set of four two-dimensional subbands labelled as `LxLy`, `LxHy`, `HxLy`, `HxHy`. The subband `LxLy` is a downsampled version of source tile that is low-pass filtered horizontally and low-pass filtered vertically. The subband `LxHy` is a downsampled version of source tile that is low-pass filtered horizontally and high-pass filtered vertically. The subband `HxLy` is a downsampled version of source tile that is high-pass filtered horizontally and low-pass filtered vertically. The subband `HxHy` is a downsampled version of source tile that is high-pass filtered horizontally and high-pass filtered vertically. The function uses the 5-3 reversible filter.

For not-in-place flavors, the subbands are stored separately in the array `pDst`. For in-place flavors, the subbands are stored in the `pSrcDstTile` (see [Figure 15-3](#))

**Figure 15-3 Subband Positions for in-place Operations**



The size of the required external buffer should be determined by calling the [ippiWTGetBufSize\\_B53\\_JPEG2K](#) function prior to using the wavelet transform function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.

`ippStsStepErr` Indicates an error condition if any of the specified step values is zero or negative.

---

## WTInv\_B53\_JPEG2K

*Performs a tile-oriented inverse wavelet transform with reversible filter.*

---

### Syntax

```
IppStatus ippWTInv_B53_JPEG2K_16s_C1R(const Ipp16s* pSrc[4],
    int srcStep[4], Ipp16s* pDst, int dstStep, const IppiRect* pTileRect,
    Ipp8u* pBuffer);

IppStatus ippWTInv_B53_JPEG2K_32s_C1R(const Ipp32s* pSrc[4],
    int srcStep[4], Ipp32s* pDst, int dstStep, const IppiRect* pTileRect,
    Ipp8u* pBuffer);

IppStatus ippWTInv_B53_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile, int
    srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);

IppStatus ippWTInv_B53_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile, int
    srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
```

### Parameters

<i>pSrc</i>	An array of pointers to the subbands of the source image (in the order LxLy, LxHy, HxLy, HxHy).
<i>srcStep</i>	Array of distances in bytes between starts of consecutive lines in the subbands of the source image.
<i>pDst</i>	Pointer to the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDstTile</i>	Pointer to the source and destination image for in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
<i>pTileRect</i>	Pointer to the tile rectangle structure.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

The function `ippiWTInv_B53_JPEG2K` is declared in the `ippj.h` file. This function performs an inverse discrete wavelet transformation using a subband reconstruction in accordance with [ISO15444], Annex F, *Discrete Wavelet Transformation of Tile-Components*. The function performs one-level reconstruction of the tile of the destination image `pDst` from the set of four two-dimensional subbands labelled as `LxLy`, `LxHy`, `HxLy`, `HxHy`. The subband `LxLy` is a downsampled version of the initial tile that is low-pass filtered horizontally and low-pass filtered vertically. The subband `LxHy` is a downsampled version of the initial tile that is low-pass filtered horizontally and high-pass filtered vertically. The subband `HxLy` is a downsampled version of the initial tile that is high-pass filtered horizontally and low-pass filtered vertically. The subband `HxHy` is a downsampled version of the initial tile that is high-pass filtered horizontally and high-pass filtered vertically. The function uses the 5-3 reversible filter.

For not-in-place flavors, the source subbands are placed separately in the array `pDst`. For in-place flavors, the source subbands are stored in the `pSrcDstTile` (see [Figure 15-3](#))

The size of the required external buffer should be determined by calling the [ippiWTGetBufSize\\_B53\\_JPEG2K](#) function prior to using the wavelet transform function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified step values is zero or negative.

---

## WTGetBufSize\_D97\_JPEG2K

*Computes the size of the buffer wavelet transform with irreversible filter*

---

## Syntax

```
IppStatus ippiWTGetBufSize_D97_JPEG2K_16s_C1R(const IppiRect* pTileRect,
int* pSize);
```

```

IppStatus ippiWTGetBufSize_D97_JPEG2K_32s_C1R(const IppiRect* pTileRect,
    int* pSize);
IppStatus ippiWTGetBufSize_D97_JPEG2K_16s_C1R(const IppiRect* pTileRect,
    int* pSize);
IppStatus ippiWTGetBufSize_D97_JPEG2K_32s_C1IR(const IppiRect* pTileRect,
    int* pSize);

```

### Parameters

<i>pTileRect</i>	Pointer to the tile rectangle structure.
<i>pSize</i>	Pointer to the computed value of the buffer.

### Description

The function `ippiWTGetBufSize_D97_JPEG2K` is declared in the `ippj.h` file. This function computes the size in bytes of the work buffer that is required for a tile-oriented irreversible wavelet transform functions [ippiWTFwd\\_D97\\_JPEG2K](#) and [ippiWTInv\\_D97\\_JPEG2K](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

---

## WTFwd\_D97\_JPEG2K

*Performs a tile-oriented forward wavelet transform with irreversible filter.*

---

### Syntax

```

IppStatus ippiWTFwd_D97_JPEG2K_16s_C1R(const Ipp16s* pSrc, int srcStep,
    const IppiRect* pTileRect, Ipp16s* pDst[4], int dstStep[4],
    Ipp8u* pBuffer);
IppStatus ippiWTFwd_D97_JPEG2K_32s_C1R(const Ipp32s* pSrc, int srcStep,
    const IppiRect* pTileRect, Ipp32s* pDst[4], int dstStep[4],
    Ipp32s* pBuffer);
IppStatus ippiWTFwd_D97_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile, int srcDstStep,
    const IppiRect* pTileRect, Ipp8u* pBuffer);

```

```
IppStatusippiWTFwd_D97_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile, int srcDstStep,
    const IppiRect* pTileRect, Ipp8u* pBuffer);
```

## Parameters

<i>pSrc</i>	Pointer to the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pTileRect</i>	Pointer to the tile rectangle structure.
<i>pDst</i>	Array of pointers to the subbands of the destination image (in the order LxLy, LxHy, HxLy, HxHy).
<i>dstStep</i>	Array of distances in bytes between starts of consecutive lines in the subbands of the destination image.
<i>pSrcDstTile</i>	Pointer to the source and destination image for in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
<i>pBuffer</i>	Pointer to the buffer.

## Description

The function `ippiWTFwd_D97_JPEG2K` is declared in the `ippj.h` file. This function performs a forward discrete wavelet transformation of the tile-component using a subband decomposition in accordance with [ISO15444], *Annex F, Discrete Wavelet Transformation of Tile-Components*. The function performs one-level decomposition by downsampling each tile of the source image *pSrc* into a set of four two-dimensional subbands *pDst* labelled as LxLy, LxHy, HxLy, HxHy. The subband LxLy is a downsampled version of the source tile to which the horizontal and vertical low-pass filters were applied. The subband LxHy is a downsampled version of the source tile to which a horizontal low-pass and vertical and high-pass filters were applied. The subband HxLy is a downsampled version of the source tile to which the horizontal and vertical high-pass filters were applied. The subband HxHy is a downsampled version of the source tile to which horizontal high-pass and vertical high-pass filters were applied. The function uses the 9-7 irreversible filter.

For not-in-place flavors, the subbands are stored separately in the array *pDst*. For in-place flavors, the subbands are stored in the *pSrcDstTile* (see [Figure 15-3](#))

The size of the required external buffer should be determined by calling the [ippiWTGetBufSize\\_D97\\_JPEG2K](#) function prior to using the wavelet transform function.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified step values is zero or negative.

---

## WTInv\_D97\_JPEG2K

*Performs a tile-oriented inverse wavelet transform with irreversible filter.*

---

### Syntax

```
IppStatus ippiWTInv_D97_JPEG2K_16s_C1R(const Ipp16s* pSrc[4],
    int srcStep[4], Ipp16s* pDst, int dstStep, const IppiRect* pTileRect,
    Ipp8u* pBuffer);

IppStatus ippiWTInv_D97_JPEG2K_32s_C1R(const Ipp32s* pSrc[4],
    int srcStep[4], Ipp32s* pDst, int dstStep, const IppiRect* pTileRect,
    Ipp8u* pBuffer);

IppStatus ippiWTInv_D97_JPEG2K_16s_C1IR(Ipp16s* pSrcDstTile,
    int srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);

IppStatus ippiWTInv_D97_JPEG2K_32s_C1IR(Ipp32s* pSrcDstTile,
    int srcDstStep, const IppiRect* pTileRect, Ipp8u* pBuffer);
```

### Parameters

<code>pSrc</code>	An array of pointers to the source subbands (in the order LxLy, LxHy, HxLy, HxHy).
<code>srcStep</code>	Array of distances in bytes between starts of consecutive lines in the the source subbands.
<code>pDst</code>	Pointer to the destination image.
<code>dstStep</code>	Distance in bytes between starts of consecutive lines in the the destination image.



<i>pSrcDstTile</i>	Pointer to the source and destination image for in-place operation.
<i>srcDstStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
<i>pTileRect</i>	Pointer to the tile rectangle structure.
<i>pBuffer</i>	Pointer to the buffer.

## Description

The function `ippiWTInv_D97_JPEG2K` is declared in the `ippj.h` file. This function performs an inverse discrete wavelet transformation using a subband reconstruction in accordance with [\[ISO15444\]](#), *Annex F, Discrete Wavelet Transformation of Tile-Components*. The function performs one-level reconstruction of the tile of the destination image *pDst* from the set of four two-dimensional subbands *pSrc* labelled as *LxLy*, *LxHy*, *HxLy*, *HxHy*. The subband *LxLy* is a downsampled version of the initial tile to which horizontal and vertical low-pass filters were applied. The subband *LxHy* is a downsampled version of the initial tile to which a horizontal low-pass and vertical high-pass filters were applied. The subband *HxLy* is a downsampled version of the initial tile to which horizontal high-pass and vertical low-pass filters were applied. The subband *HxHy* is a downsampled version of the initial tile to which horizontal and vertical high-pass filters were applied. The function uses the 9-7 irreversible filter.

For not-in-place flavors, the source subbands are placed separately in the array *pDst*. For in-place flavors, the source subbands are placed in the *pSrcDstTile* (see [Figure 15-3](#))

The size of the required external buffer should be determined by calling the [ippiWTGetBufSize\\_D97\\_JPEG2K](#) function prior to using the wavelet transform function.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of ROI has zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified step values is zero or negative.

## JPEG2000 Entropy Coding and Decoding Functions

This section describes the image processing functions that perform arithmetic encoding and decoding procedures and are specific for JPEG 2000 image coding system. These functions are listed in the [Table 15-11](#).

**Table 15-11 Entropy Coder Functions**

Function Base Name	Description
<a href="#">EncodeInitAlloc JPEG2K</a>	Allocates memory and initializes an entropy encoder state structure
<a href="#">EncodeFree JPEG2K</a>	Deallocates memory allocated for an entropy encoding state structure
<a href="#">EncodeLoadCodeBlock JPEG2K</a>	Loads code block and prepares data for entropy encoding
<a href="#">EncodeStoreBits JPEG2K</a>	Produces output codestream
<a href="#">EncodeGetTermPassLen JPEG2K</a>	Returns the length of the specified terminated coding pass.
<a href="#">EncodeGetRate JPEG2K</a>	Returns estimated target bit rate for the specified coding pass
<a href="#">EncodeGetDist JPEG2K</a>	Returns estimated distortion of the image for the specified coding pass
<a href="#">DecodeGetBufSize JPEG2K</a>	Computes the size of the working buffer for decoding routine
<a href="#">DecodeCodeBlock JPEG2K</a>	Decodes the compressed code block data
<b>Decoder Functions for Progressive Mode</b>	
<a href="#">DecodeCBProgrGetStateSize JPEG2K</a>	Computes the size of the external buffer for the decoder state structure.
<a href="#">DecodeCBProgrInit JPEG2K</a>	Initializes the decoder state structure in the external buffer/
<a href="#">DecodeCBProgrInitAlloc JPEG2K</a>	Allocates memory and initializes the decoder state structure in the external buffer.
<a href="#">DecodeCBProgrFree JPEG2K</a>	Frees memory allocated for the decoder state structure.
<a href="#">DecodeCBProgrAttach JPEG2K</a>	Attaches the buffer with data for progressive decoding.
<a href="#">DecodeCBProgrSetPassCounter JPEG2K</a>	Sets the value of internal coding pass counter
<a href="#">DecodeCBProgrGetPassCounter JPEG2K</a>	Returns the value of the internal coding pass counter.
<a href="#">DecodeCBProgrGetCurBitPlane JPEG2K</a>	Returns the number of the current bit plane

Table 15-11    Entropy Coder Functions (continued)

Function Base Name	Description
<a href="#">DecodeCBProgrStep_JPEG2K</a>	Performs a single step of decoding.

The implemented adaptive arithmetic encoding is performed as two successive procedures. First, the prepared code block is pre-processed by the function `ippiEncodeLoadCodeBlock_JPEG2K`. After that, the function `ippiEncodeStoreBits_JPEG2K` performs final stages of encoding and produces the output codestream. In the course of encoding, the state structure `pState` is used, which is allocated and initialized by the function `ippiEncodeInitAlloc_JPEG2K`.

## EncodeInitAlloc\_JPEG2K

*Allocates memory and initializes an entropy encoder state structure.*

### Syntax

```
IppStatus ippiEncodeInitAlloc_JPEG2K(IppiEncodeState_JPEG2K** pState,
                                     IppiSize codeBlockMaxSize);
```

### Parameters

- `pState`

Pointer to variable that returns the pointer to allocated and initialized encoder state structure.
- `codeBlockMaxSize`

Maximum size of code block in pixels.

### Description

The function `ippiEncodeInitAlloc_JPEG2K` is declared in the `ippj.h` file. This function allocates and initializes the entropy encoder state structure that is used in the encoding procedures.

### Return Values

- `ippStsNoErr`

Indicates no error.
- `ippStsSizeErr`

Indicates an error condition if the width or height of the code block has zero or negative value.
- `ippStsMemAllocErr`

Indicates an error condition if memory allocation fails.

---

## EncodeFree\_JPEG2K

*Frees memory allocated for an entropy encoding state structure.*

---

### Syntax

```
IppStatus ippiEncodeFree_JPEG2K(IppiEncodeState_JPEG2K* pState);
```

### Parameters

`pState`                      Pointer to the allocated and initialized encoder state structure.

### Description

The function `ippiEncodeFree_JPEG2K` is declared in the `ippj.h` file. This function frees memory allocated for the entropy encoder state structure by the function `ippiEncodeInitAlloc_JPEG2K`.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pState</code> pointer to a state structure is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid state structure is passed.

---

## EncodeLoadCodeBlock\_JPEG2K

*Loads a code block and prepares data for entropy encoding.*

---

### Syntax

```
IppStatus ippiEncodeLoadCodeBlock_JPEG2K_32s_C1R(const Ipp32s* pSrc,  
    int srcStep, IppiSize codeBlockSize, IppiEncodeState_JPEG2K* pState,  
    IppiWTSubband subband, int magnBits, IppiMQTermination mqTermType,  
    IppiMQRateAppr mqRateAppr, int codeStyleFlags, int* pSfBits,  
    int* pNoFPasses, int* pNoFTermPasses);
```

Parameters

<i>pSrc</i>	Pointer to the source code block.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the code block.
<i>codeBlockSize</i>	Size of the code block.
<i>pState</i>	Pointer to allocated and initialized encoder state structure.
<i>subband</i>	Wavelet transform subband contained the given code block. Possible values are listed in the <a href="#">Table 15-12</a> .
<i>magnBits</i>	Non-fractional bit count in integer representation.
<i>codeStyleFlags</i>	Specifies the options for encoding procedure. The possible values are listed in the <a href="#">Table 15-13</a> .
<i>mqTermType</i>	Termination mode for MQ-coder. Possible values are listed in the <a href="#">Table 15-14</a> .
<i>mqRateAppr</i>	Specifies the bit rate estimation model. Only one value, <code>ippMQRateApprGood</code> , that sets the non-optimal approximation model is available now.
<i>pSfBits</i>	Pointer to the variable that returns the number of significant bit planes.
<i>pNOFPasses</i>	Pointer to the variable that returns the number of coding passes.
<i>pNOFTermPasses</i>	Pointer to the variable that returns the number of terminated coding passes.

Description

The function `ippiEncodeLoadCodeBlock_JPEG2K` is declared in the `ippj.h` file. This function performs the first encoding procedure, that is, loads the specified code block, computes required parameters of the MQ-coder, and prepares data for the second encoding procedure. The wavelet transform subband should be specified in the *subband* parameter. Possible values are listed in the [Table 15-12](#).

Table 15-12 Subband Parameter for the JPEG2000 Entropy Coder Functions

Value	Description
<code>ippWTSubbandLxLy</code>	Subband obtained by low-pass filtering along x and y directions
<code>ippWTSubbandLxHy</code>	Subband obtained by low-pass filtering along x and high-pass filtering along y direction

**Table 15-12 Subband Parameter for the JPEG2000 Entropy Coder Functions (continued)**

Value	Description
ippWTSubbandHxLy	Subband obtained by high-pass filtering along x and low-pass filtering along y direction
ippWTSubbandHxHy	Subband obtained by high-pass filtering along x and y directions

**Table 15-13 CodeStyleFlag Argument for the JPEG2000 Entropy Coder Functions**

Value	Description
IPP_JPEG2K_VERTICALLY_CAUSAL_CONTEXT	Sets the creation of vertically causal context
IPP_JPEG2K_SELECTIVE_MQ_BYPASS	Sets the selective MQ encoding bypass, which is raw encoding for some coding passes
IPP_JPEG2K_TERMINATE_ON_EVERY_PASS	Sets the termination of MQ coder after every coding pass
IPP_JPEG2K_RESETCTX_ON_EVERY_PASS	Sets the reset of MQ coder context after each coding pass
IPP_JPEG2K_USE_SEGMENTATION_SYMBOLS	Uses the segmentation symbol sequence for error resilience
IPP_JPEG2K_LOSSLESS_MODE	Indicates that the lossless wavelet transforms are used in the rate-distortion estimation
IPP_JPEG2K_DEC_CONCEAL_ERRORS	Allows the error concealment in the last bit-plane where the errors were detected
IPP_JPEG2K_DEC_DO_NOT_CLEAR_CB	Does not clear the code block data before decoding
IPP_JPEG2K_DEC_DO_NOT_RESET_LOW_BITS	Does not reset low-order bits after decoding
IPP_JPEG2K_DEC_DO_NOT_CLEAR_SFBUFFER	Does not clear the buffer before decoding to keep the previous significance state
IPP_JPEG2K_DEC_CHECK_PRED_TERM	Verifies during decoding if the predictable termination is correct; if not - "damage in codeblock" error code is generated. It should be used in predictable termination mode.

**Table 15-14 MqTermType Argument for the JPEG2000 Entropy Coder Functions**

Value	Description
ippMQTermSimple	Simple termination - some extra bytes are added
ippMQTermNearOptimal	Near optimal termination mode
ippMQTermPredictable	Termination mode for predictable error resilience

The variable `sfBits` returns the number of significant bit planes only. For example, if all source pixels are non-negative and their maximum value is 0xA (binary 1010), then `sfBits` should return 4 significant bits. The higher bits are not coded.

Only significant non-fractional bits in integer representation are coded.

`magnBits` specifies the number of non-fractional bits. For example, if `magnBits` = 11, the 20 (31-11) least significant bits will not be coded.

Negative integers in the code block should be presented in direct code with the sign in the most significant bit (31<sup>st</sup> in the zero-based numeration, when the least significant bit has a zero index). This code may be effectively obtained using the specially designed function [ippiComplement](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of the code block has zero or negative value.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid state structure is passed.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> has zero or negative value.

---

## EncodeStoreBits\_JPEG2K

*Produces output codestream.*

---

### Syntax

```
ippStatus ippiEncodeStoreBits_JPEG2K_1u(Ipp8u* pDst, int* pDstLen,
    IppiEncodeState_JPEG2K* pState, int* pIsNotFinish);
```

### Parameters

<code>pDst</code>	Pointer to the destination data buffer.
<code>pDstLen</code>	Pointer to the destination buffer length.
<code>pState</code>	Pointer to the allocated and initialized encoder state structure.

*pIsNotFinish* Pointer to the variable indicating that the encoding is completed.

### Description

The function `ippiEncodeStoreBits_JPEG2K` is declared in the `ippj.h` file. This function performs the second encoding procedure and produces the output codestream.

### Application Notes

The parameter *dstLen* is used both for reading and writing. It should be passed to the function with a valid buffer length, and then it will return the length of used (filled) buffer.

[Example 15-1](#) shows how this function may be used.

---

#### Example 15-1 Using the `ippiEncodeStoreBits_JPEG2K` Function

---

```
int isNotFinish = 1;
while(isNotFinish)
{
    int len = BUFFER_LEN;
    ippiEncodeStoreBits_JPEG2K_1u(buffer, &len, state, &isNotFinish);
    // You can write compressed data to the output stream, for example:
    // fwrite(buffer, sizeof(Ipp8u), len, file);
}
```

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of the destination buffer has zero or negative value.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid state structure is passed.



---

## EncodeGetTermPassLen\_JPEG2K

Returns the length of the specified terminated coding pass.

---

### Syntax

```
IppStatus ippiEncodeGetTermPassLen_JPEG2K(IppiEncodeState_JPEG2K* pState,  
int passNumber, int* pPassLen);
```

### Parameters

<i>pState</i>	Pointer to the allocated and initialized encoder state structure.
<i>passNumber</i>	Number of the terminated coding pass.
<i>pPassLen</i>	Pointer to the variable that returns the length of the terminated coding pass.

### Description

The function `ippiEncodeGetTermPassLen_JPEG2K` is declared in the `ippj.h` file. This function returns the length *pPassLen* of the terminated coding pass of a given number *passNumber* that is stored in the *pState* state structure.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid state structure is passed.
<code>ippStsJPEG2KBadPassNumber</code>	Indicates an error condition if <i>passNumber</i> exceeds the allowed boundaries.

---

## EncodeGetRate\_JPEG2K

*Returns estimated bit rate for the specified coding pass.*

---

### Syntax

```
IppStatus ippiEncodeGetRate_JPEG2K(IppiEncodeState_JPEG2K* pState,  
    int passNumber, int* pRate);
```

### Parameters

<i>pState</i>	Pointer to the allocated and initialized encoder state structure.
<i>passNumber</i>	Number of the specified coding pass.
<i>pRate</i>	Pointer to the variable that returns the estimated rate for the specified coding pass.

### Description

The function `ippiEncodeGetRate_JPEG2K` is declared in the `ippj.h` file. This function returns the estimated target bit rate *pRate* for the coding pass of a given number *passNumber*. The estimated bit rate is computed during the encoding procedure and then stored in the *pState* state structure. Therefore it is available only after completing both encoding procedures.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid state structure is passed.
<code>ippStsJPEG2KBadPassNumber</code>	Indicates an error condition if <i>passNumber</i> exceeds the allowed boundaries.

---

## EncodeGetDist\_JPEG2K

*Returns estimated distortion of the image for the specified coding pass.*

---

### Syntax

```
IppStatus ippiEncodeGetDist_JPEG2K(IppiEncodeState_JPEG2K* pState,  
    int passNumber, Ipp64f* pDist);
```

### Parameters

<i>pState</i>	Pointer to the allocated and initialized encoder state structure.
<i>passNumber</i>	Number of the specified coding pass.
<i>pDist</i>	Pointer to the variable that returns the value of the estimated distortion.

### Description

The function `ippiEncodeGetDist_JPEG2K` is declared in the `ippj.h` file. This function returns the cumulative weighted mean square error (MSE) distortion *pDist* of the reconstructed image for the coding pass of a given number *passNumber*. The distortions are computed in accordance with [ISO15444], Annex J, section J.14, Rate Control] during the encoding procedure and then stored in the *pState* structure. Therefore, it is available only after completing both encoding procedures.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to an invalid state structure is passed.
<code>ippStsJPEG2KBadPassNumber</code>	Indicates an error condition if <i>passNumber</i> exceeds the allowed boundaries.

---

## DecodeGetBufSize\_JPEG2K

*Computes the size of the work buffer for decoding routine.*

---

### Syntax

```
IppStatus ippidecodeGetBufSize_JPEG2K(IppiSize codeBlockSize,  
    int* pSize);
```

### Parameters

<i>codeBlockSize</i>	Maximum size of code block in pixels.
<i>pSize</i>	Pointer to the variable that returns the size of the work buffer.

### Description

The function `ippidecodeGetBufSize_JPEG2K` is declared in the `ippj.h` file. This function computes the size (in bytes) of the work buffer that corresponds to the specified maximum size of a code block (in pixels).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of the code block has zero or negative value.

---

## DecodeCodeBlock\_JPEG2K

*Decodes the compressed code block data.*

---

### Syntax

```
IppStatus ippidecodeCodeBlock_JPEG2K_1u32s_C1R(const Ipp8u* pSrc,  
    Ipp32s* pDst, int dstStep, IppiSize codeBlockSize, IppiWTSubband subband,  
    int sfBits, int nOfPasses, const int* pTermPassLen, int nOfTermPasses,  
    int codeStyleFlags, int* pErrorBitPlane, Ipp8u* pBuffer);
```

## Parameters

<i>pSrc</i>	Pointer to the source compressed code block.
<i>pDst</i>	Pointer to the destination for decoded data.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination buffer.
<i>codeBlockSize</i>	Size of the code block.
<i>subband</i>	Wavelet transform subband contained the given code block. Possible values are listed in the <a href="#">Table 15-12</a>
<i>sfBits</i>	Number of significant bit planes.
<i>nOfPasses</i>	Number of coding passes.
<i>pTermPassLen</i>	Pointer to the array that contains the length of each terminated coding pass.
<i>nOfTermPasses</i>	Number of terminated coding passes.
<i>codeStyleFlags</i>	Options for the decoding procedure. Possible values are listed in the <a href="#">Table 15-13</a> .
<i>pErrorBitPlane</i>	Pointer to the bit plane that contains the first error returned when the damaged code block occurs.
<i>pBuffer</i>	Pointer to the work buffer.

## Description

The function `ippiDecodeCodeBlock_JPEG2K` is declared in the `ippj.h` file. This function decodes the compressed code block *pSrc* of data using adaptive arithmetic decoding procedure and stores the decoded data in the *pDst* buffer.

Negative integers in the code block should be presented in the direct code with the sign in the most significant bit (31<sup>st</sup> in the zero-based numeration, when the least significant bit has a zero index). This code may be effectively obtained using the specially designed function [ippiComplement](#).




---

**NOTE.** *errorBitPlane* should be set to `NULL` if the corresponding information is not required.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if the width or height of the code block has zero or negative value.
<code>ippStsJPEG2KDamagedCodeBlock</code>	Indicates an error condition if a code block contains damaged data.
<code>ippStsStepErr</code>	Indicates an error condition if <i>dstStep</i> has zero or negative value.

---

## DecodeCBProgrGetStateSize\_JPEG2K

*Computes the size of the external buffer for the decoder state structure.*

---

### Syntax

```
IppStatus ippidecodeCBProgrGetStateSize_JPEG2K(IppiSize codeBlockSize,  
int* pStateSize);
```

### Parameters

<i>codeBlockSize</i>	Maximum size of the codeblock.
<i>pStateSize</i>	Pointer to the variable that returns the size of the buffer for the decoder state structure.

### Description

The function `ippidecodeCBProgrGetStateSize_JPEG2K` is declared in the `ippj.h` file. This function computes the size of the external buffer for the decoder state structure. This parameter is depending on the maximum size of the codeblock *codeBlockSize*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pStateSize</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>codeBlockSize</i> has a field with zero or negative value.

---

## DecodeCBProgrInit\_JPEG2K

*Initializes the decoder state structure in the external buffer.*

---

### Syntax

```
IppStatus ippidecodeCBProgrInit_JPEG2K(IppiDecodeCBProgrState_JPEG2K* pState);
```

### Parameters

*pState*                      Pointer to the decoder state structure.

### Description

The function `ippidecodeCBProgrInit_JPEG2K` is declared in the `ippj.h` file. This function initialize the decoder state structure for progressive mode *pState* in the external buffer which size should be computed previously by calling the function [ippidecodeCBProgrGetStateSize\\_JPEG2K](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to the invalid structure is passed.

---

## DecodeCBProgrInitAlloc\_JPEG2K

*Allocates memory and initializes the decoder state structure in the external buffer.*

---

### Syntax

```
IppStatus ippidecodeCBProgrInitAlloc_JPEG2K(IppiDecodeCBProgrState_JPEG2K**  
ppState, IppiSize codeBlockMaxSize);
```

### Parameters

<i>ppState</i>	Pointer to the pointer to the decoder state structure.
<i>codeBlockMaxSize</i>	Maximum size of the codeblock.

### Description

The function `ippiDecodeCBProgrInitAlloc_JPEG2K` is declared in the `ippj.h` file. This function allocates memory and initialize the decoder state structure for progressive mode *pState*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>codeBlockMaxSize</i> has a field with zero or negative value.

---

## DecodeCBProgrFree\_JPEG2K

*Frees memory allocated for the decoder state structure.*

---

### Syntax

```
IppStatus ippiDecodeCBProgrFree_JPEG2K(IppiDecodeCBProgrState_JPEG2K* pState);
```

### Parameters

<i>pState</i>	Pointer to the allocated and initialized decoder state structure.
---------------	---

### Description

The function `ippiDecodeCBProgrFree_JPEG2K` is declared in the `ippj.h` file. This function frees memory allocated by the function [DecodeCBProgrInitAlloc\\_JPEG2K](#) for the decoder state structure.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> pointer is NULL.



`ippStsContextMatchErr` Indicates an error condition if a pointer to the invalid structure is passed.

---

## DecodeCBProgrAttach\_JPEG2K

*Attaches the buffer with data for progressive decoding.*

---

### Syntax

```
IppStatus ippidecodeCBProgrAttach_JPEG2K_32s_C1R(Ipp32s* pDst, int dstStep,
IppiSize codeBlockSize, IppiDecodeCBProgrState_JPEG2K* pState, IppiWTSubband
subband, int sfBits, int codeStyleFlags);
```

### Parameters

<code>pDst</code>	Pointer to the destination buffer.
<code>dstStep</code>	Step in bytes through the destination buffer.
<code>codeBlockSize</code>	Size of the codeblock.
<code>pState</code>	Pointer to the decoder state structure.
<code>subband</code>	Wavelet transform subband contained the given code block. Possible values are listed in the <a href="#">Table 15-12</a> .
<code>sfBits</code>	Number of significant bits in code-block.
<code>codeStyleFlags</code>	Specifies the options for encoding procedure. The possible values are listed in the <a href="#">Table 15-13</a> .

### Description

The function `ippidecodeCBProgrAttach_JPEG2K` is declared in the `ippj.h` file. This function attaches a destination buffer `pDst` for the code-block rectangle and fix its parameters to perform the progressive decoding.



---

**CAUTION.** The image data buffer should not be moved in memory and should not be freed during whole decoding procedure, that is until the last step of decoding progression and damage correction will be completed.

---

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if the value of <i>dstStep</i> is zero or negative.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to the invalid structure is passed.

---

## DecodeCBProgrSetPassCounter\_JPEG2K

*Sets the value of internal coding pass counter.*

---

### Syntax

```
IppStatus ippiDecodeCBProgrSetPassCounter_JPEG2K(int nOfPasses,  
IppiDecodeCBProgrState_JPEG2K* pState);
```

### Parameters

<i>nOfPasses</i>	Number of coding passes.
<i>pState</i>	Pointer to the decode state structure.

### Description

The function `ippiDecodeCBProgrSetPassCounter_JPEG2K` is declared in the `ippj.h` file. This function sets the value of internal coding pass counter.

The single coding pass is “sub-atom” for coding progression, but their counts in a single terminated segment (for example, the segment filled to the byte boundary) is available from the JPEG 2000 stream or file. During decoding step, pass counter is decremented by one or some other number depending on coding options. If pass counter is set to zero, decoding process will be blocked internally. Externally the pass counter value can be obtained by calling `ippiDecodeCodeBlockProgrGetPassCounter_JPEG2K` function.

The coding pass counter value can be set before any progression step as well as before series of steps.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to the invalid structure is passed.

---

## DecodeCBProgrGetPassCounter\_JPEG2K

*Returns the value of the internal coding pass counter.*

---

### Syntax

```
IppStatus ippDecodeCBProgrGetPassCounter_JPEG2K(IppiDecodeCBProgrState_JPEG2K*  
pState, int* pNoOfResidualPasses);
```

### Parameters

*pState* Pointer to the decode state structure.

*pNoOfResidualPasses* Pointer to the number of residual coding passes.

### Description

The function `ippDecodeCBProgrGetPassCounter_JPEG2K` is declared in the `ippj.h` file. This function returns the value of internal coding pass counter that is received from the decoding state structure *pState*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to the invalid structure is passed.

---

## DecodeCBProgrGetCurBitPlane\_JPEG2K

*Returns the number of the current bit plane.*

---

### Syntax

IppStatus

```
ippiDecodeCBProgrGetCurBitPlaneNum_JPEG2K(IppiDecodeCBProgrState_JPEG2K*  
pState, int* pBitPlaneNum);
```

### Parameters

<i>pState</i>	Pointer to the decode state structure.
<i>pBitPlaneNum</i>	Pointer to the variable containing current bit plane number.

### Description

The function `ippiDecodeCBProgrGetCurBitPlane_JPEG2K` is declared in the `ippj.h` file. This function returns the current bit plane number. Initially the bit plane counter is set to  $(sfBits - 1)$  value (see the function [ippiDecodeCBProgrAttach\\_JPEG2K](#)). During successful decoding step, the bit plane counter is decremented by one or some other number depending on coding options.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if <i>pState</i> pointer is NULL.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to the invalid structure is passed.

---

## DecodeCBProgrStep\_JPEG2K

*Performs a single step of decoding.*

---

### Syntax

```
IppStatus ippDecodeCBProgrStep_JPEG2K(const Ipp8u* pSrc, int srcLen,  
    IppDecodeCBProgrState_JPEG2K* pState);
```

### Parameters

<i>pSrc</i>	Pointer to the source data.
<i>srcLen</i>	Length of the segment.
<i>pState</i>	Pointer to the decode state structure.

### Description

The function `ippDecodeCBProgrStep_JPEG2K` is declared in the `ippj.h` file. This function performs one step of the decoding progression.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error if one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error if length of terminated segment is specified incorrectly.
<code>ippStsContextMatchErr</code>	Indicates an error condition if a pointer to the invalid structure is passed.

## Component Transform Functions

This section describes the functions that perform component transformation specific for JPEG 2000 image coding system. All functions implement the component transformations in accordance with its definition by [ISO15444], *Annex G, DC Level Shifting and Multiple Component Transformations.*]

[Table 15-15](#). lists these functions described in more detail later in this section.

**Table 15-15 Component Transform Functions**

Function Base Name	Description
<a href="#">RCTFwd_JPEG2K</a>	Performs forward reversible component transformation.
<a href="#">RCTInv_JPEG2K</a>	Performs inverse reversible component transformation.
<a href="#">ICTFwd_JPEG2K</a>	Performs forward irreversible component transformation.
<a href="#">ICTInv_JPEG2K</a>	Performs inverse irreversible component transformation.

---

### RCTFwd\_JPEG2K

*Performs forward reversible component transformation.*

---

#### Syntax

##### Case 1: Operation on pixel-order data

```
IppStatus ippiRCTFwd_JPEG2K_32s_C3P3R(const Ipp32s* pSrc, int srcStep,
    Ipp32s* pDst[3], int dstStep, IppiSize roiSize);
```

##### Case 2: In-place operation on planar data

```
IppStatus ippiRCTFwd_JPEG2K_32s_P3IR(Ipp32s* pSrcDst[3], int srcDstStep,
    IppiSize roiSize);
```

#### Parameters

<i>pSrc</i>	Pointer to the source image ROI.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the array of pointers to ROIs in the planes of the destination image.

<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the array of pointers to the source and destination image ROIs for the in-place operation.
<i>dstSrcStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiRCTFwd_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs forward reversible component transformation (RCT) of the image buffer *pSrc*. The RCT is applied to all image component samples  $I_0$ ,  $I_1$ ,  $I_2$ , corresponding to the first, second, and third components, and produces the transform samples  $Y_0$ ,  $Y_1$ ,  $Y_2$  in accordance with the following formulas:

$$Y_0 = \left\lfloor \frac{I_0 + 2 I_1 + I_2}{4} \right\rfloor$$

$$Y_1 = I_2 - I_1$$

$$Y_2 = I_0 - I_1$$

where the notation  $\lfloor x \rfloor$  designates the floor function, that is, the largest integer not exceeding  $x$ .

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## RCTInv\_JPEG2K

*Performs inverse reversible component transformation.*

---

### Syntax

#### Case 1: Operation on planar data

```
IppStatus ippiRCTInv_JPEG2K_32s_P3C3R(const Ipp32s* pSrc[3], int srcStep,  
    Ipp32s* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: In-place operation on planar data

```
IppStatus ippiRCTInv_JPEG2K_32s_P3IR(Ipp32s* pSrcDst[3], int srcDstStep,  
    IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the array of pointers to the ROIs in the planes of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the array of pointers to ROIs in the planes of the source and destination image for the in-place operation.
<i>dstSrcStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippiRCTFwd_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs inverse RCT of the image buffer *pSrc* after inverse wavelet transformation. The inverse RCT is applied to transformed image component samples  $Y_0, Y_1, Y_2$  and produces the corresponding samples  $I_0, I_1, I_2$  in accordance with the following formulas:

where the notation  $\lfloor x \rfloor$  designates the floor function, that is, the largest integer not exceeding  $x$ .



$$I_1 = Y_0 - \left\lfloor \frac{Y_2 + Y_1}{4} \right\rfloor$$

$$I_0 = Y_2 + I_1$$

$$I_2 = Y_1 + I_1$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

---

## ICTFwd\_JPEG2K

*Performs forward irreversible component transformation.*

---

### Syntax

#### Case 1: Operation on pixel-order data

```
IppStatus ippICTFwd_JPEG2K_32f_C3P3R(const Ipp32f* pSrc, int srcStep,
    Ipp32f* pDst[3], int dstStep, IppiSize roiSize);
```

#### Case 2: In-place operation on planar data

```
IppStatus ippICTFwd_JPEG2K_<mod>(Ipp<datatype>* pSrcDst[3], int
    srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

`16s_P3IR`    `32s_P3IR`    `32f_P3IR`

### Parameters

*pSrc*                      Pointer to the source image ROI.

<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the array of pointers to ROIs in the planes of the destination image.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the array of pointers to the source and destination image ROIs for the in-place operation.
<i>dstSrcStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

## Description

The function `ippiICTFwd_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs forward irreversible component transformation (ICT) of the image buffer *pSrc*. The ICT is applied to all image component samples  $I_0$ ,  $I_1$ ,  $I_2$ , corresponding to the first, second, and third components, and produces the transform samples  $Y_0$ ,  $Y_1$ ,  $Y_2$  in accordance with the following formulas:

$$\begin{aligned}
 Y_0 &= 0.299 * I_0 + 0.587 * I_1 + 0.114 * I_2 \\
 Y_1 &= -0.16875 * I_0 - 0.33126 * I_1 + 0.5 * I_2 \\
 Y_2 &= 0.5 * I_0 - 0.41869 * I_1 - 0.08131 * I_2
 \end{aligned}$$




---

**NOTE.** If the first three components are Red, Green and Blue components, then the forward ICT is of a YCbCr transformation.

---

## Return Values

<code>ippiStsNoErr</code>	Indicates no error.
<code>ippiStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippiStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippiStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

## ICTInv\_JPEG2K

*Performs inverse irreversible component transformation.*

### Syntax

#### Case 1: Operation on planar data

```
IppStatus ippICTInv_JPEG2K_32f_P3C3R(const Ipp32f* pSrc[3], int
    srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

#### Case 2: In-place operation on planar data

```
IppStatus ippICTInv_JPEG2K_<mod>(Ipp<datatype>* pSrcDst[3], int
    srcDstStep, IppiSize roiSize);
```

Supported values for *mod*:

```
16s_P3IR    32s_P3IR    32f_P3IR
```

### Parameters

<i>pSrc</i>	Pointer to the array of pointers to the ROIs in the planes of the source image.
<i>srcStep</i>	Distance in bytes between starts of consecutive lines in the source image.
<i>pDst</i>	Pointer to the destination image ROI.
<i>dstStep</i>	Distance in bytes between starts of consecutive lines in the destination image.
<i>pSrcDst</i>	Pointer to the array of pointers to ROIs in the planes of the source and destination image for the in-place operation.
<i>dstSrcStep</i>	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
<i>roiSize</i>	Size of the source and destination ROI in pixels.

### Description

The function `ippICTFwd_JPEG2K` is declared in the `ippj.h` file. It operates with ROI (see [Regions of Interest in Intel IPP](#)).

This function performs inverse irreversible component transformation (ICT) of the image buffer *pSrc* after inverse wavelet transformation. The inverse ICT is applied to transformed image component samples  $Y_0, Y_1, Y_2$  and produces the corresponding samples  $I_0, I_1, I_2$  in accordance with the following formulas:

$$\begin{aligned} I_0 &= Y_0 + 1.402 * Y_2 \\ I_1 &= Y_0 - 0.34413 * Y_1 - 0.71414 * Y_2 \\ I_2 &= Y_0 + 1.772 * Y_1 \end{aligned}$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when any of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roiSize</i> has a field with zero or negative value.
<code>ippStsStepErr</code>	Indicates an error condition if any of the specified buffer step values is zero or negative.

This chapter describes the subset of Intel IPP functions designed for video coding applications. This subset is a library of functions for encoding and decoding of video data according to MPEG-1 ([ISO11172]), MPEG-2 ([ISO13818]), MPEG-4 ([ISO14496A]), DV ([IEC61834]), H.261 ([ITUH261]), H.263 ([ITUH263]), and H.264 ([JVTG050] and [ITUH264]) standards. These functions have a convenient interface and present an appropriate solution for both encoding and decoding pipeline and, as all other parts of Intel IPP, are intended for the development of high-performance cross-platform code.

In addition to common data types used in Intel IPP for image processing (see [Data Types](#) in Chapter 2), video coding functions subset defines its specific data type:

**Table 16-1**      **Conventional vs Intel IPP Data Types**

Bit Depth	Conventional Type	Intel IPP Type
32	signed int	Ipp32s
16	signed short	Ipp16s
8	signed chart	Ipp8s
32	unsigned int	Ipp32u
16	unsigned short	Ipp16u
8	unsigned chart	Ipp8u

Descriptors in the names of video coding functions (see [Function Naming](#) in Chapter 2) may indicate the standard according to which the function operates (MPEG1, MPEG2, MPEG4, H263, H264, DV) or the size of a block of elements processed together (16x16, 16x8, 8x16, 8x8, 8x4). The first number is the width of the block and the second is its height. For functions that process one channel the descriptor C1 is used. In-place function names contain the descriptor I, while functions are not in-place by default.

The use of video coding functions is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm> .

Descriptions of Intel IPP video coding functions are grouped into the following sections:

- [General Functions](#)
- [MPEG-1 and MPEG-2](#)
- [DV](#)
- [MPEG-4](#)
- [H.261](#)
- [H.263](#)
- [H.264](#)

## General Functions

These functions are quite universal and can be used in different codecs. This category includes functions of the following types:

- [Variable Length Decoding](#) functions that use Variable Length Code tables to decode video data.
- [Motion Compensation](#) functions that calculate sum of the residual block and the predicted block for reconstruction of the source block.
- [Motion Estimation](#) functions that calculate:
  - residual block - difference between source block and predicted block
  - some characteristics of the residual block
  - some characteristics of the source block. These characteristics can be used for comparison of the blocks.
- [Scanning Functions](#) that convert two-dimensional arrays into one-dimensional data and vice versa using the predefined scan pattern.
- [Color Conversion](#) functions that convert images with rotation of the destination image.
- [Video Processing](#) function that is used to process decompressed video data.

## Structures and Enumerators

The structure `IppMotionVector` can be used in both H.263 and MPEG-4 video processing functions and is defined as follows:

```
typedef struct {
    Ipp16s dx;
    Ipp16s dy;
} IppMotionVector;
```

The following enumerator indicates the type of a decoded macroblock or a macroblock to be encoded. `IPPVC_MB_STUFFING` indicates that bit stuffing codeword was decoded from the bitstream or should be encoded into the bitstream.

```
enum
{
    IPPVC_MBTYPETYPE_INTER = 0,      /* valid in P-picture or P-VOP */
    IPPVC_MBTYPETYPE_INTER_Q = 1,     /* P-picture or P-VOP */
    IPPVC_MBTYPETYPE_INTER4V = 2,     /* P-picture or P-VOP */
    IPPVC_MBTYPETYPE_INTRA = 3,       /* I- and P-picture, or I- and P-VOP */
    IPPVC_MBTYPETYPE_INTRA_Q = 4,     /* I- and P-picture, or I- and P-VOP */
    IPPVC_MBTYPETYPE_INTER4V_Q = 5,   /* P-picture or P-VOP */
    IPPVC_MBTYPETYPE_DIRECT = 6,      /* B-picture or B-VOP */
    IPPVC_MBTYPETYPE_INTERPOLATE = 7, /* B-picture or B-VOP */
    IPPVC_MBTYPETYPE_BACKWARD = 8,    /* B-picture or B-VOP */
    IPPVC_MBTYPETYPE_FORWARD = 9,     /* B-picture or B-VOP */
    IPPVC_MB_STUFFING = 255           /* indicates bit stuffing codeword */
};
```

The following enumerator indicates the type of scan performed on DCT coefficients:

```
enum
{
    IPPVC_SCAN_NONE = -1,             no scan to be performed
    IPPVC_SCAN_ZIGZAG = 0,            classical zigzag scan
    IPPVC_SCAN_VERTICAL = 1,          alternate vertical scan
    IPPVC_SCAN_HORIZONTAL = 2         alternate horizontal scan
};
```

See typical scan patterns in [Figure 16-12](#) and the scanning process according to the classical scanning pattern in [Figure 16-17](#).

The following enumerator indicates the type of interpolation performed on a picture. This enumerator is used, for example, in spatial scalability mode.

```
enum
{
    IPPVC_INTERP_NONE = 0,          no interpolation
    IPPVC_INTERP_HORIZONTAL = 1,    1-D horizontal interpolation
    IPPVC_INTERP_VERTICAL = 2,      1-D vertical interpolation
    IPPVC_INTERP_2D = 3             2-D interpolation
};
```

The following enumerator indicates the type of rotate operation for general color conversion functions:

```
enum {
    IPPVC_ROTATE_DISABLE = 0,
    IPPVC_ROTATE_90CCW = 1,
    IPPVC_ROTATE_90CW = 2,
    IPPVC_ROTATE_180 = 3
};
```

The following enumerator indicates the type of color space conversion for general color conversion functions:

```
enum
{
    IPPVC_CbYCr422ToBGR565 = 0,
    IPPVC_CbYCr422ToBGR555 = 1
};
```

The enumerator `IPPVC_MC_APX` indicates the type of prediction for motion compensation or motion estimation.

```
typedef enum _IPPVC_MC_APX{
    IPPVC_MC_APX_FF = 0x0,
    IPPVC_MC_APX_FH = 0x4,
    IPPVC_MC_APX_HF = 0x8,
    IPPVC_MC_APX_HH = 0x0c
} IPPVC_MC_APX;
```

The first descriptors `FF`, `FH`, `HF`, `HH` show whether prediction is accurate to full pel (F) or to half a pel (H). The first letter indicates the accuracy in horizontal direction and the second — in vertical direction.

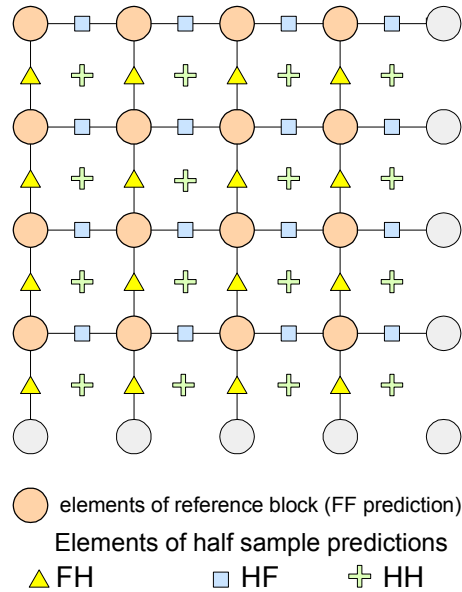
Most of general functions use parameter *mcType* (see enumeration [IPPVC\\_MC\\_APX](#)). It is used for calculating prediction on the basis of block in the reference frame. See [Figure 16-1](#).

- If *mcType* = `IPPVC_MC_APX_FF`, the reference block is used as the prediction block.



- If  $mcType = IPPVC\_MC\_APX\_FH$  or  $IPPVC\_MC\_APX\_HF$ , each element of the prediction block is calculated as average of two elements of the reference block.
- If  $mcType = IPPVC\_MC\_APX\_HH$ , each element of the prediction block is calculated as average of four elements of the reference block.

**Figure 16-1 Predictions for 4x4 Reference Block**



### UV Block

$UV$  block of size  $(2*H) \times W$  combines  $U$  block of size  $H \times W$  and  $V$  block of size  $H \times W$ . The formula for  $UV$  is as follows:

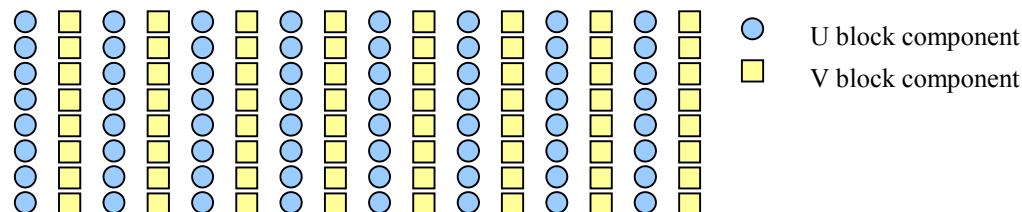
$$UV[2*i + 1, j] = V[i, j]$$

$$UV[2*i, j] = U[i, j],$$

where  $i=[0, H-1]$  and  $j=[0, W-1]$ . See the figure below for an example of a  $UV$  block.

**Figure 16-2 16x8 UV Block (8x8 U Block and 8x8 V Block)**

---



**Table 16-2 General Functions**

---

Function Short Name	Description
<b>Variable Length Decoding</b>	
<a href="#">HuffmanTableInitAlloc</a>	Allocates memory and initializes structure for table with one-to-one code/value correspondence.
<a href="#">HuffmanRunLevelTableInitAlloc</a>	Allocates memory and initializes structure for table with one-to-two code/value correspondence.
<a href="#">DecodeHuffmanOne</a>	Decodes one code using specified table and gets one decoded value.
<a href="#">DecodeHuffmanPair</a>	Decodes one code using specified table and gets two decoded values.
<a href="#">HuffmanTableFree</a>	Frees memory allocated for VLC table.
<b>Motion Compensation</b>	
<a href="#">MC16x16</a>	Performs motion compensation for a predicted 16X16 block.
<a href="#">MC16x8</a>	Performs motion compensation for a predicted 16X8 block.
<a href="#">MC8x16</a>	Performs motion compensation for a predicted 8X16 block.
<a href="#">MC8x8</a>	Performs motion compensation for a predicted 8X8 block.
<a href="#">MC8x4</a>	Performs motion compensation for a predicted 8X4 block.

**Table 16-2**      **General Functions (continued)**

Function Short Name	Description
<a href="#">MC4x8</a>	Performs motion compensation for a predicted 4X8 block.
<a href="#">MC4x4</a>	Performs motion compensation for a predicted 4X4 block.
<a href="#">MC2x4</a>	Performs motion compensation for a predicted 2X4 block.
<a href="#">MC4x2</a>	Performs motion compensation for a predicted 4X2 block.
<a href="#">MC2x2</a>	Performs motion compensation for a predicted 2X2 block.
<a href="#">MC16x4</a>	Performs motion compensation for predicted UV 16X4 block.
<a href="#">MC16x8UV</a>	Performs motion compensation for predicted UV 16X8 block.
<a href="#">MC16x16B</a>	Performs motion compensation for a bi-predicted 16X16 block.
<a href="#">MC16x8B</a>	Performs motion compensation for a bi-predicted 16X8 block.
<a href="#">MC8x16B</a>	Performs motion compensation for a bi-predicted 8X16 block.
<a href="#">MC8x8B</a>	Performs motion compensation for a bi-predicted 8X8 block.
<a href="#">MC8x4B</a>	Performs motion compensation for a bi-predicted 8X4 block.
<a href="#">MC4x8B</a>	Performs motion compensation for a bi-predicted 4X8 block.
<a href="#">MC4x4B</a>	Performs motion compensation for a bi-predicted 4X4 block.
<a href="#">MC2x4B</a>	Performs motion compensation for a bi-predicted 2X4 block.
<a href="#">MC4x2B</a>	Performs motion compensation for a bi-predicted 4X2 block.
<a href="#">MC2x2B</a>	Performs motion compensation for a bi-predicted 2X2 block.

**Table 16-2      General Functions (continued)**

Function Short Name	Description
<a href="#">MC16x4B</a>	Performs motion compensation for bi-predicted UV 16X4 block.
<a href="#">MC16x8UVB</a>	Performs motion compensation for bi-predicted UV 16X8 block.
<a href="#">Copy8x8, Copy16x16</a>	Copy the fixed size block to the destination block.
<a href="#">Copy8x4HP, Copy8x8HP, Copy16x8HP, Copy16x16HP</a>	Copy fixed size blocks with half-pixel accuracy.
<a href="#">InterpolateAverage8x4, InterpolateAverage8x8, InterpolateAverage16x8, InterpolateAverage16x16</a>	Interpolate source block according to half-pixel offset and average the result with destination block.
<a href="#">Add8x8</a>	Adds two blocks with saturation.
<a href="#">Add8x8HP</a>	Adds blocks interpolated with half-pixel accuracy prediction to difference with saturation.
<a href="#">AddC8x8</a>	Adds a constant to 8x8 block with saturation.
<a href="#">Average8x8, Average16x16</a>	Average two blocks.
<b>Motion Estimation</b>	
<b>Difference Evaluation</b>	
<a href="#">GetDiff16x16</a>	Evaluates difference between current predicted and reference blocks of 16x16 elements.
<a href="#">GetDiff16x8</a>	Evaluates difference between current predicted and reference blocks of 16x8 elements.
<a href="#">GetDiff8x8</a>	Evaluates difference between current predicted and reference blocks of 8x8 elements.
<a href="#">GetDiff8x16</a>	Evaluates difference between current predicted and reference blocks of 8x16 elements.
<a href="#">GetDiff8x4</a>	Evaluates difference between current predicted and reference blocks of 8x4 elements.
<a href="#">GetDiff4x4</a>	Evaluates difference between current predicted and reference 4x4 blocks.
<a href="#">GetDiff16x16B</a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 16x16 elements.
<a href="#">GetDiff16x8B</a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 16x8 elements.

**Table 16-2      General Functions (continued)**

Function Short Name	Description
<a href="#"><u>GetDiff8x8B</u></a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x8 elements.
<a href="#"><u>GetDiff8x16B</u></a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x16 elements.
<a href="#"><u>GetDiff8x4B</u></a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x4 elements.
<a href="#"><u>Sub8x8, Sub16x16</u></a>	Subtract two blocks and store the result in the third block.
<a href="#"><u>SubSAD8x8</u></a>	Subtracts two block, stores the result in the third block and computes a sum of absolute differences.
<b>Sum of Squares of Differences Evaluation</b>	
<a href="#"><u>SqrDiff16x16</u></a>	Evaluates sum of squares of differences between current and reference blocks.
<a href="#"><u>SqrDiff16x16B</u></a>	Evaluates sum of squares of differences between the current bi-predicted block and the mean of two reference blocks.
<a href="#"><u>SSD8x8</u></a>	Evaluates sum of squares of differences between current and reference 8X8 blocks.
<a href="#"><u>SSD4x4</u></a>	Evaluates sum of squares of differences between current and reference 4X4 blocks.
<b>Block Variance and Mean Evaluation</b>	
<a href="#"><u>VarMean8x8</u></a>	Evaluates variance and mean of a 8X8 block of unsigned char or short integer values.
<b>Evaluation of Variances and Means of Blocks of Difference Between Two Blocks</b>	
<a href="#"><u>VarMeanDiff16x16</u></a>	Evaluates variances and means of four 8x8 blocks of difference between two 16x16 blocks.
<a href="#"><u>VarMeanDiff16x8</u></a>	Evaluates variances and means of four 8x8 blocks of difference between two 16x8 blocks.
<b>Block Variance Evaluation</b>	
<a href="#"><u>Variance16x16</u></a>	Evaluates variance of current block.
<b>Evaluation of Block Deviation</b>	
<a href="#"><u>MeanAbsDev16x16</u></a>	Evaluates mean absolute deviation for a 16x16 block.
<b>Edges Detection</b>	
<a href="#"><u>EdgesDetect16x16</u></a>	Detects edges inside a 16X16 block.

**Table 16-2      General Functions (continued)**

Function Short Name	Description
<b>SAD Functions</b>	
<a href="#"><u>SAD16x16</u></a>	Evaluates sum of absolute difference between current and reference blocks.
<a href="#"><u>SAD8x8</u></a>	Evaluates sum of absolute difference between current and reference 8X8 blocks.
<a href="#"><u>SAD4x4</u></a>	Evaluates sum of absolute difference between current and reference 4X4 blocks.
<a href="#"><u>SAD16x16Blocks8x8</u></a>	Evaluates four partial sums of absolute differences between current and reference 16X16 blocks.
<a href="#"><u>SAD16x16Blocks4x4</u></a>	Evaluates 16 partial sums of absolute differences between current and reference 16X16 blocks.
<a href="#"><u>FrameFieldSAD16x16</u></a>	Calculates SAD between field lines and SAD between frame lines of block 16x16.
<b>Sum of Differences Evaluation</b>	
<a href="#"><u>SumsDiff16x16Blocks4x4</u></a>	Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 16X16 blocks.
<a href="#"><u>SumsDiff8x8Blocks4x4</u></a>	Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 8X8 blocks.
<b>Scanning</b>	
<a href="#"><u>ScanInv</u></a>	Performs classical zigzag, alternate-horizontal, or alternate-vertical inverse scan on a block stored in a compact buffer.
<a href="#"><u>ScanFwd</u></a>	Performs classical zigzag, alternate-horizontal, or alternate-vertical forward scan on a block.
<a href="#"><u>ZigzagInvClassical Compact,</u></a> <a href="#"><u>ZigzagInvHorizontal Compact,</u></a> <a href="#"><u>ZigzagInvVertical Compact</u></a>	Perform classical, horizontal, or vertical inverse zigzag scan on a block stored in a compact buffer.
<b>Color Conversion</b>	
<a href="#"><u>CbYCr422ToYCbCr420 Rotate</u></a>	Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image with rotation.
<a href="#"><u>ResizeCCRotate</u></a>	Creates a low-resolution preview image for high-resolution video or still capture applications.

Table 16-2      General Functions (continued)

Function Short Name	Description
Video Processing	
<a href="#">DeinterlaceFilterTriangle</a>	Deiterlaces video plane.

Variable Length Decoding

Video data in the bit stream is encoded with Variable Length Code (VLC) tables so that the shortest codes correspond to the most frequent values and the longer codes correspond to the less frequent values. Tables for each video standard contain specific possible codes and their values.

VLC decoding functions can work with two types of tables:

- 1. Tables, in which one code corresponds to one value (see [Table 16-3](#) for an example)
- 2. Tables, in which one code corresponds to two values (see [Table 16-4](#) for an example).

Table 16-3      One-to-One Code/Value Correspondence

Code	Value
0001 1	-3
0011	-2
011	-1
1	0
010	1
0010	2
0001 0	3

Table 16-4      One-to-Two Code/Value Correspondence

Code	Value 1	Value 2
0001 1	1	0
0011	1	1
011	1	2
1	2	0

Table 16-4 One-to-Two Code/Value Correspondence (continued)

Code	Value 1	Value 2
010	2	2
0010	3	2
0001 0	4	1

Video data VLC-decoding starts with memory allocation and table initialization. Then the decoding proper should be made and the allocated memory must be released.

Table 16-5 VLC Decoding Functions

One-to-One Correspondence	One-to-Two Correspondence	Short Description
<a href="#">HuffmanTableInitAlloc</a>	<a href="#">HuffmanRunLevelTableInitAlloc</a>	Allocates memory and initializes structure that is used for decoding.
<a href="#">DecodeHuffmanOne</a>	<a href="#">DecodeHuffmanPair</a>	Decodes one code using specified table.
	<a href="#">HuffmanTableFree</a>	Frees memory allocated for VLC table.

For using `ippiVCHuffmanInitAlloc_32s` and `ippiVCHuffmanInitAllocRL_32s` functions, the source table should have the following structure:

Example 16-1 Source Table Structure for `HuffmanTableInitAlloc` Functions

```
static Ipp32s Table[] =
{
    max_bits,           // The maximum length of code
    total_subt,         // The total number of all subtables
    sub_sz1,            // The sizes of subtables. Their sum must
    sub_sz2,            // be equal to the maximum length of code
    ...,               // max_bits and their number should be
    N1,                // equal to total_subt.
    Code1, Value11, [Value21] // The number of 1-bit codes
    Code2, Value12, [Value22] // The 1-bit codes
    ...,              //
    N2,                // The number of 2-bit codes
    Code1, Value11, [Value21] //
    Code2, Value12, [Value22] // The 2-bit codes
    ...,              //
    Nm,                // The number of max_bits, -bit codes
    Code1, Value11, [Value21] //
```



Example 16-1 Source Table Structure for HuffmanTableInitAlloc Functions (continued)

```
Code2, Value12, [Value22] // max_bits codes
...//
-1 // The significant value to indicate the
// end of table
};
```

Notes:

1. The values Value2<sub>i</sub> are used for the second type table (one-to-two code/value correspondence) and must be in the range [-32768, 32767] or [0, 65535].
2. The values Value1<sub>i</sub> must be in the range:
  - [-8388608, 8388607] or [0, 16777215] in the case of first type table (one-to-one correspondence)
  - [-128, 127] or [0, 255] in the case of second type table (one-to-two correspondence).

Using Subtables

Subtables are used for large source tables. Internally, VLC decoding is done by table lookups. A straightforward approach would be to have one lookup of a lookup table of size 2<sup>L</sup>, where L is the maximum number of code bits. This approach however is proven to be memory inefficient. Therefore, subtables are used to balance the memory and the lookup speed. Using subtables reduces the memory size but increases the number of lookups. The greater is the number of subtables, the smaller is the structure size but the number of decoding operations increases. Table structure of this kind provides optimal ratio between the table redundancy and the number of decoding operations. Each subtable works on some numbers of bits out of L. In `ippiVCHuffmanInitAlloc_32s` and `ippiVCHuffmanInitAllocRL_32s` functions subtables sizes are specified manually. Any combinations of subtables are possible as long as their sum adds up to L.

Consider an example of VLC decoding. [Table 16-6](#) gives codes and their values.

Table 16-6 Example of VLC Decoding

Code	Value
0	A
1100	B

Table 16-6 Example of VLC Decoding (continued)

Code	Value
101	C
1110	D
100	E
11110	F
11111	G
1101	H

First, let us consider decoding when subtables are not used. In this case the source table should look as follows:

```
static Ipp32s Table[] =
{
    5, // The maximum length of code
    1, // The total number of all subtables
    5, // The size of 1 subtable
    1, // The number of 1-bit codes
    0/*0*/,A, // code1, value1
    0, // The number of 2-bit codes
    2 // The number of 3-bit codes
    5/*101*/,C, // code1, value1
    4/*100*/,E, // code2, value2
    3 // The number of 4-bit codes
    12/*1100*/,B, // code1, value1
    14/*1110*/,D, // code2, value2
    13/*1101*/,H, // code3, value3
    2 // The number of 4-bit codes
    30/*11110*/,F, // code1, value1
    31/*11111*/,G, // code2, value2
    -1
};
```

The function `ippiVCHuffmanInitAlloc_32s` is used to form a structure that contains the following data:

Table 16-7 Structure Data For a Case With One Subtable

Code (length=5)	Value	Length
00000	A	1
00000	A	1
...	...	

Table 16-7      Structure Data For a Case With One Subtable (continued)

Code (length=5)	Value	Length
01111	A	1
10100	C	3
10101	C	3
10110	C	3
10111	C	3
10000	E	3
10000	E	3
10000	E	3
10000	E	3
11000	B	4
11001	B	4
11100	D	4
11101	D	4
11010	H	4
11011	H	4
11110	F	5
11111	G	5

In process of decoding, 5 bits are extracted from the bitstream. They are used to find a corresponding code in [Table 16-7](#) and matching value and length. (5-Length) bits are returned to the bitstream.

Now, consider an example when two subtables are used for decoding. The source table structure will look as follows:

```
static Ipp32s Table[]=
{
5,                                // The maximum length of code
2,                                // The total number of all subtables
3,                                // The size of subtable 1
2,                                // The size of subtable 2
1,                                // The number of 1-bit codes
0/*0*/,A,                        // code1, value1
0,                                // The number of 2-bit codes
2,                                // The number of 3-bit codes
5/*101*/,C,                      // code1, value1
4/*100*/,E,                      // code2, value2
3,                                // The number of 4-bit codes
12/*1100*/,B,                   // code1, value1
```

```

14/*1110*/,D,          // code2, value2
13/*1101*/,H,          // code3, value3
2                      // The number of 4-bit codes
30/*11110*/,F,         // code1, value1
31/*11111*/,G,         // code2, value2
-1
};

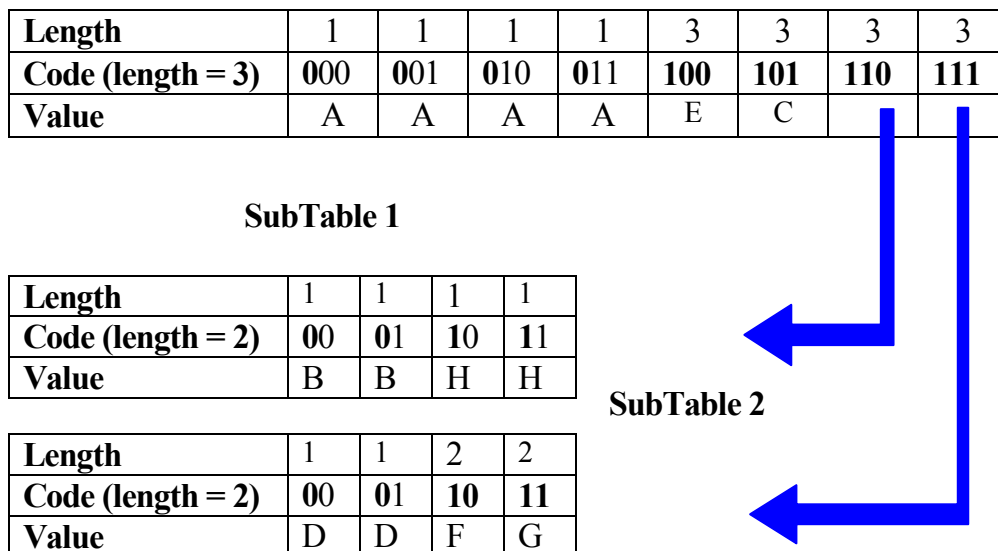
```

---

The function `ippiVCHuffmanInitAlloc_32s` is used to form a structure that contains two subtables:

**Figure 16-3 VLC Subtables**

---



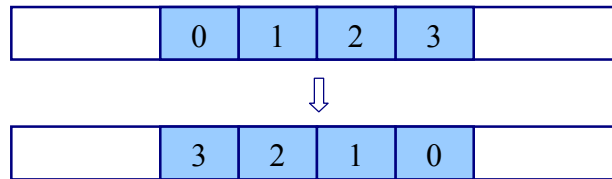
Subtable 1 contains the values with the code length that does not exceed 3. Subtable 2 contains the values with the code length that exceeds 3, with the corresponding code consisting of 3-bit code from Subtable 1 and 2-bit code from Subtable 2.

In process of decoding 3 bits are extracted from the bitstream. They are used to find a corresponding code in Subtable 1. The matching value and length are found if the code is other than 100 or 111. (3-Length) bits are returned to the bitstream. Otherwise, two more bits are extracted from the bitstream to find respective value and length in Subtable 2 and (2-Length) bits are returned to the butstream afterwards.

### `ppBitStream` and `pOffset` Parameters

Before decoding and applying the Intel IPP functions, all bytes in each 32-bit double word in the bit stream must be flipped ([Figure 16-4](#)) to improve performance of decoding functions.

**Figure 16-4** Flipping of Bytes



The parameters `ppBitStream` and `pOffset` define start position for a subsequent code: current element of the array and position in this element. These parameters are updated by function.

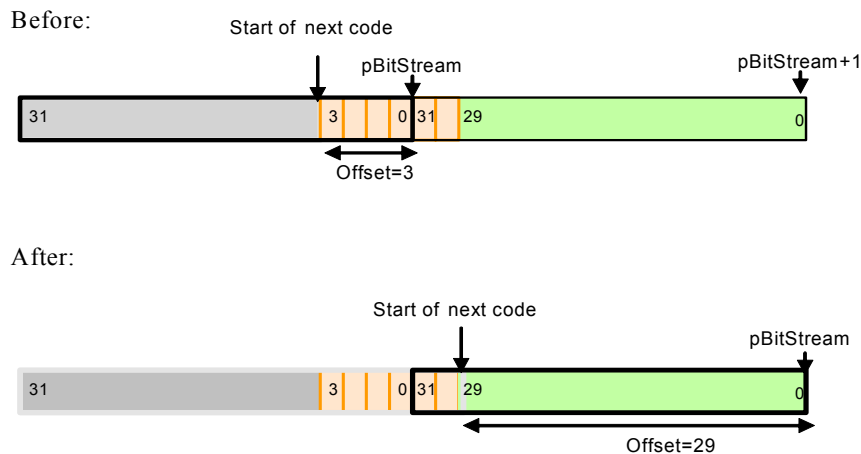
`ppBitStream` Double pointer to the current position in the bitstream.

`pOffset` Pointer to the offset between the bit that `*ppBitStream` points to and the start of the code.

See [Figure 16-5](#) for an example of getting one code from the bitstream.

**Figure 16-5 Getting One Code From Bitstream**

---



The VLC functions are used in the decoders included into IPP Samples. See [introduction](#) to the chapter.

## HuffmanTableInitAlloc

*Allocates memory and initializes structure for table with one-to-one code/value correspondence.*

---

### Syntax

```
IppStatus ippiHuffmanTableInitAlloc_32s(const Ipp32s* pSrcTable,
    IppVCHuffmanSpec_32s** ppDstSpec);
```

### Parameters

*pSrcTable*            Pointer to the source table (see [Source Table Structure](#)).

*ppDstSpec*            Double pointer to the destination decoding table.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanTableInitAlloc_32s` allocates memory and initializes structure for a table in which one code corresponds to one value (See [Table 16-3](#)). This structure is used for decoding.

See [Table 16-11](#) for details of specific use of the function in MPEG1, MPEG2 standards.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsMemAllocErr</code>	Indicates an error when no memory is allocated.

---

## HuffmanRunLevelTableInitAlloc

*Allocates memory and initializes structure for table with one-to-two code/value correspondence.*

---

## Syntax

```
IppStatus ippiHuffmanRunLevelTableInitAlloc_32s(const Ipp32s* pSrcTable,
        IppVCHuffmanSpec_32s** ppDstSpec);
```

<code>pSrcTable</code>	Pointer to the source table (see <a href="#">Source Table Structure</a> ).
<code>ppDstSpec</code>	Double pointer to the destination decoding table.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanRunLevelTableInitAlloc_32s` allocates memory and initializes structure for a table in which one code corresponds to two value (See [Table 16-4](#)). This structure is used for decoding.

See [Table 16-11](#) for details of specific use of the function in MPEG1, MPEG2 standards.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

`ippStsMemAllocErr` Indicates an error when no memory is allocated.

## Decoding with non-Run-Level Tables

This function decodes all codes except for DCT coefficients.

---

## DecodeHuffmanOne

*Decodes one code using specified table and gets one decoded value.*

---

### Syntax

```
IppStatus ippDecodeHuffmanOne_1u32s(Ipp32u** ppBitStream, Ipp32s* pOffset,
    Ipp32s* pDst, const IppVCHuffmanSpec_32s *pDecodeTable);
```

### Parameters

`ppBitStream` Double pointer to the current position in the bitstream.

`pOffset` Pointer to offset between the bit that `ppBitStream` points to and the start of the code. `pOffset` may vary from zero to 31.

`pDst` Pointer to the destination result, that is, one value.

`pDecodeTable` Pointer to the decoding table.

### Description

This function is declared in the `ippvc.h` header file. The function `ippDecodeHuffmanOne_1u32s` decodes one code from the bitstream using a specified table, sends the result (one value) to destination data, and resets the pointers to new positions (See example in [Figure 16-5](#)). The function uses the table derived by the [HuffmanTableInitAlloc](#) function.

### Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error when at least one input pointer is NULL.

`ippStsH263VLCCodeErr` Indicates an error in process of decoding.



---

## DecodeHuffmanPair

*Decodes one code using specified table and gets two decoded values.*

---

### Syntax

```
IppStatus ippiDecodeHuffmanPair_1u16s(Ipp32u **ppBitStream, Ipp32s *pOffset,  
    const IppVCHuffmanSpec_32s *pDecodeTable, Ipp8s *pFirst, Ipp16s *pSecond);
```

### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bitstream.
<i>pOffset</i>	Pointer to offset between the bit that <i>**ppBitStream</i> points to and the start of the code. <i>pOffset</i> may vary from zero to 31.
<i>pDecodeTable</i>	Pointer to the decoding table.
<i>pFirst</i>	Pointer to the first destination value.
<i>pSecond</i>	Pointer to the second destination value.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiDecodeHuffmanPair_1u16s` decodes one code from the bitstream using a specified table, sends the result (two values) to destination data, and resets the pointers to new positions (See example in [Figure 16-5](#)). The function uses the table derived by the [HuffmanRunLevelTableInitAlloc](#) function.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsH263VLCCodeErr</code>	Indicates an error in process of decoding.

## Memory Release

---

### HuffmanTableFree

*Frees memory allocated for VLC table.*

---

#### Syntax

```
IppStatus ippiHuffmanTableFree_32s(IppVCHuffmanSpec_32s **ppDecodeTable);
```

#### Parameters

*ppDecodeTable*            Double pointer to the decoding table.

#### Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanTableFree_32s` frees memory at *ppDecodeTable* allocated for VLC table.

#### Return Values

`ippStsNoErr`               Indicates no error.

`ippStsNullPtrErr`       Indicates an error when at least one input pointer is NULL.

## Motion Compensation

These functions calculate sum of the residual block and the predicted block for reconstruction of the source block. After processing, the data is converted with saturation from `Ipp16s` to `Ipp8u` type.

Motion compensation functions are divided into two groups:

- Functions for predicted block, using one reference block for prediction
- Functions for bi-predicted block, using two reference blocks for prediction; in this case predictions are first calculated for each reference block, and then prediction is calculated as average of two predictions.

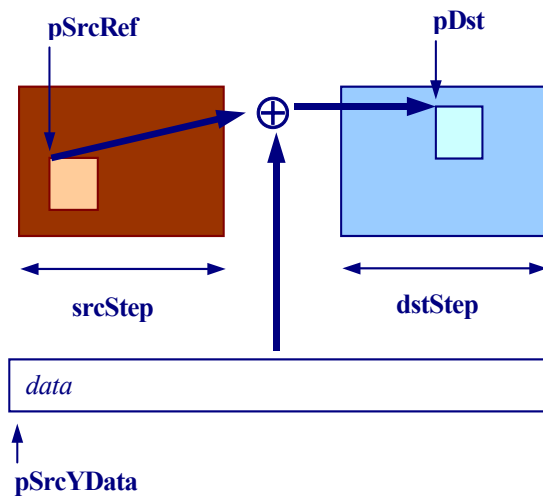
The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from

<http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm> .

### Predicted Blocks

These functions use one reference block for prediction.

**Figure 16-6 Motion Compensation Scheme for Predicted Block**



## MC16x16

*Performs motion compensation for predicted 16X16 block.*

---

### Syntax

```
IppStatus ippIMC16x16_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC MC APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC16x16_8u_C1` calculates sum of 16x16 residual block and 16x16 predicted block for reconstruction of the source block (See [Figure 16-6](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC MC APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## MC16x8

*Performs motion compensation for predicted 16X8 block.*

---

### Syntax

```
IppStatus ippMC16x8_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC MC APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippMC16x8_8u_C1` calculates sum of 16x8 residual block and 16x8 predicted block for reconstruction of the source block (See [Figure 16-6](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC MC APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC8x16

*Performs motion compensation for predicted 8X16 block.*

---

### Syntax

```
IppStatus ippIMC8x16_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC MC APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC8x16_8u_C1` calculates sum of 8x16 residual block and 8x16 predicted block for reconstruction of the source block (See [Figure 16-6](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC MC APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## MC8x8

*Performs motion compensation for predicted 8X8 block.*

---

### Syntax

```
IppStatus ippIMC8x8_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC8x8_8u_C1` calculates sum of 8x8 residual block and 8x8 predicted block for reconstruction of the source block (See [Figure 16-6](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC8x4

*Performs motion compensation for predicted 8X4 block.*

---

### Syntax

```
IppStatus ippIMC8x4_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s  
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s mcType,  
    Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC8x4_8u_C1` calculates sum of 8x4 residual block and 8x4 predicted block for reconstruction of the source block (See [Figure 16-6](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.



## MC4x8

*Performs motion compensation for predicted 4X8 block.*

### Syntax

```
IppStatus ippIMC4x8_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC4x8_8u_C1` calculates sum of 4x8 residual block and 4x8 predicted block for reconstruction of the source block (See [Figure 16-6](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC4x4

*Performs motion compensation for predicted 4X4 block.*

---

### Syntax

```
IppStatus ippIMC4x4_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s  
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s mcType,  
    Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC4x4_8u_C1` calculates sum of 4x4 residual block and 4x4 predicted block for reconstruction of the source block (See [Figure 16-6](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## MC2x4

*Performs motion compensation for predicted 2X4 block.*

---

### Syntax

```
IppStatus ippIMC2x4_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC2x4_8u_C1` calculates sum of 2x4 residual block and 2x4 predicted block for reconstruction of the source block (See [Figure 16-6](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC4x2

*Performs motion compensation for predicted 4X2 block.*

---

### Syntax

```
IppStatus ippIMC4x2_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s  
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s mcType,  
    Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC4x2_8u_C1` calculates sum of 4x2 residual block and 4x2 predicted block for reconstruction of the source block (See [Figure 16-6](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## MC2x2

*Performs motion compensation for predicted 2X2 block.*

---

### Syntax

```
IppStatus ippIMC2x2_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC2x2_8u_C1` calculates sum of 2x2 residual block and 2x2 predicted block for reconstruction of the source block (See [Figure 16-6](#)). Prediction is calculated on the basis of the reference block and *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC16x4

*Performs motion compensation for predicted UV 16X4 block.*

---

### Syntax

```
IppStatus ippIMC16x4_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC MC APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC16x4_8u_C1` calculates sum of 16x4 residual UV block and 16x4 predicted UV block for reconstruction of the source UV block (See [Figure 16-2](#)). Prediction is calculated on the basis of the reference UV block and *mcType* (see [IPPVC MC APX](#)).

Half sample prediction is calculated taking into account structure of [UV block](#). So (horizontally) neighboring elements of U-block (or V-block) have indexes *i* and *i*+2 in UV block.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC16x8UV

*Performs motion compensation for predicted UV 16X8 block.*

### Syntax

```
IppStatus ippIMC16x8UV_8u_C1(const Ipp8u *pSrcRef, Ipp32s srcStep, const Ipp16s
    *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s mcType,
    Ipp32s roundControl);
```

### Parameters

<i>pSrcRef</i>	Pointer to the reference intra block.
<i>srcStep</i>	Size of the row in bytes, specifying the aligned reference frame width.
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>mcType</i>	MC type <a href="#">IPPVC MC APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippIMC16x8UV_8u_C1` calculates sum of 16x8 residual UV block and 16x8 predicted UV block for reconstruction of the source UV block (See [Figure 16-2](#)). Prediction is calculated on the basis of the reference UV block and *mcType* (see [IPPVC MC APX](#)).

Half sample prediction is calculated taking into account structure of [UV block](#). So (horizontally) neighboring elements of U-block (or V-block) have indexes *i* and *i*+2 in UV block.

### Return Values

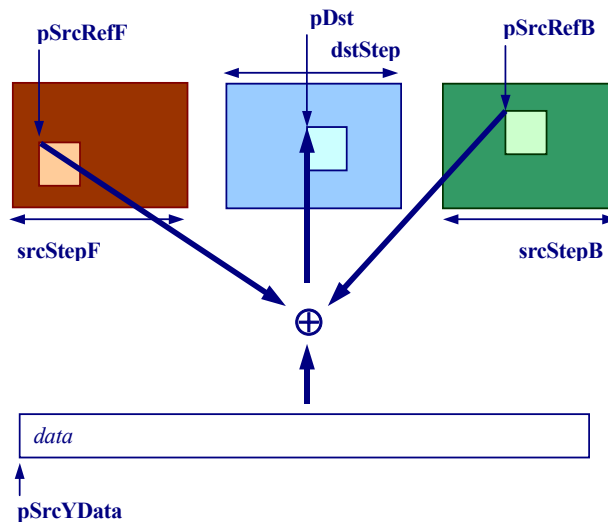
<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## Bi-Predicted Blocks

These functions use two reference blocks for prediction. Predictions are calculated for each reference block, and then prediction is calculated as average of two predictions. The prediction calculated as a rounded average of two blocks is the first to be added to the data.

**Figure 16-7 Motion Compensation Scheme for Bi-Predicted Block**

---




---

## MC16x16B

*Performs motion compensation for bi-predicted block.*

---

### Syntax

```
IppStatus ippIMC16x16B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```



## Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of the row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC16x16B_8u_C1` calculates sum of 16x16 residual block and 16x16 predicted block, which is calculated as average of two 16x16 predictions (F-prediction and B-prediction) (See [Figure 16-7](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC MC APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC16x8B

*Performs motion compensation for bi-predicted 16X8 block.*

---

### Syntax

```
IppStatus ippiMC16x8B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

### Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of the row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiMC16x8B_8u_C1` calculates sum of 16x8 residual block and 16x8 predicted block, which is calculated as average of two 16x8 predictions (F-prediction and B-prediction) (See [Figure 16-7](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC MC APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## MC8x16B

*Performs motion compensation for bi-predicted 8X16 block.*

---

### Syntax

```
IppStatus ippIMC8x16B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

### Parameters

<code>pSrcRefF</code>	Pointer to the forward reference block.
<code>srcStepF</code>	Size of the row in bytes, specifying the aligned forward reference frame width.
<code>mcTypeF</code>	Forward MC type <a href="#">IPPVC MC APX</a> .
<code>pSrcRefB</code>	Pointer to the backward reference block.
<code>srcStepB</code>	Size of the row in bytes, specifying the aligned backward reference frame width.
<code>mcTypeB</code>	Backward MC type <a href="#">IPPVC MC APX</a> .
<code>pSrcYData</code>	Pointer to the data obtained after inverse DCT.
<code>srcYDataStep</code>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<code>pDst</code>	Pointer to the destination predicted block.
<code>dstStep</code>	Size of the row in bytes, specifying the aligned destination frame width.
<code>roundControl</code>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC8x16B_8u_C1` calculates sum of 8x16 residual block and 8x16 predicted block, which is calculated as average of two 8x16 predictions (F-prediction and B-prediction) (See [Figure 16-7](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## MC8x8B

*Performs motion compensation for bi-predicted 8X8 block.*

---

## Syntax

```
IppStatus ippiMC8x8B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s roundControl);
```

## Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of the row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.

<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiMC8x8B_8u_C1` calculates sum of 8x8 residual block and 8x8 predicted block, which is calculated as average of two 8x8 predictions (F-prediction and B-prediction) (See [Figure 16-7](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC MC APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## MC8x4B

*Performs motion compensation for bi-predicted 8X4 block.*

---

### Syntax

```
IppStatus ippiMC8x4B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s  
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const  
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s  
    roundControl);
```

### Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of row in bytes, specifying the aligned backward reference frame width.

<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiMC8x4B_8u_C1` calculates sum of 8x4 residual block and 8x4 predicted block, which is calculated as average of two 8x4 predictions (F-prediction and B-prediction) (See [Figure 16-7](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## MC4x8B

*Performs motion compensation for bi-predicted 4X8 block.*

---

### Syntax

```
IppStatus ippiMC4x8B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

### Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
-----------------	---

---

<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC4x8B_8u_C1` calculates sum of 4x8 residual block and 4x8 predicted block, which is calculated as average of two 4x8 predictions (F-prediction and B-prediction) (See [Figure 16-7](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC MC APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## MC4x4B

*Performs motion compensation for bi-predicted 4X4 block.*

---

### Syntax

```
IppStatus ippiMC4x4B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s roundControl);
```

### Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiMC4x4B_8u_C1` calculates sum of 4x4 residual block and 4x4 predicted block, which is calculated as average of two 4x4 predictions (F-prediction and B-prediction) (See [Figure 16-7](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC MC APX](#)).



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## MC2x4B

*Performs motion compensation for bi-predicted 2X4 block.*

---

### Syntax

```
IppStatus ippIMC2x4B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

### Parameters

<code>pSrcRefF</code>	Pointer to the forward reference block.
<code>srcStepF</code>	Size of the row in bytes, specifying the aligned forward reference frame width.
<code>mcTypeF</code>	Forward MC type <a href="#">IPPVC MC APX</a> .
<code>pSrcRefB</code>	Pointer to the backward reference block.
<code>srcStepB</code>	Size of row in bytes, specifying the aligned backward reference frame width.
<code>mcTypeB</code>	Backward MC type <a href="#">IPPVC MC APX</a> .
<code>pSrcYData</code>	Pointer to the data obtained after inverse DCT.
<code>srcYDataStep</code>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<code>pDst</code>	Pointer to the destination predicted block.
<code>dstStep</code>	Size of the row in bytes, specifying the aligned destination frame width.
<code>roundControl</code>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC2x4B_8u_C1` calculates sum of 2x4 residual block and 2x4 predicted block, which is calculated as average of two 2x4 predictions (F-prediction and B-prediction) (See [Figure 16-7](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## MC4x2B

*Performs motion compensation for bi-predicted 4X2 block.*

---

## Syntax

```
IppStatus ippiMC4x2B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s roundControl);
```

## Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.

<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC4x2B_8u_C1` calculates sum of 4x2 residual block and 4x2 predicted block, which is calculated as average of two 4x2 predictions (F-prediction and B-prediction) (See [Figure 16-7](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC MC APX](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## MC2x2B

*Performs motion compensation for bi-predicted 2X2 block.*

---

## Syntax

```
IppStatus ippiMC2x2B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

## Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of row in bytes, specifying the aligned backward reference frame width.

<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiMC2x2B_8u_C1` calculates sum of 2x2 residual block and 2x2 predicted block, which is calculated as average of two 2x2 predictions (F-prediction and B-prediction) (See [Figure 16-7](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC\\_MC\\_APX](#)).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## MC16x4B

*Performs motion compensation for bi-predicted UV 16X4 block.*

---

### Syntax

```
IppStatus ippiMC16x4B_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

### Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
-----------------	---

---

<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of the row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiMC16x4B_8u_C1` calculates sum of 16x4 residual UV block and 16x4 predicted UV block, which is calculated as average of two 16x4 predictions (F-prediction and B-prediction) (See [Figure 16-2](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC MC APX](#)).

Half sample prediction is calculated taking into account structure of [UV block](#). So (horizontally) neighboring elements of U-block (or V-block) have indexes *i* and *i*+2 in UV block.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## MC16x8UVB

*Performs motion compensation for bi-predicted 16X8 block.*

---

### Syntax

```
IppStatus ippiMC16x8BUV_8u_C1(const Ipp8u *pSrcRefF, Ipp32s srcStepF, Ipp32s
    mcTypeF, const Ipp8u *pSrcRefB, Ipp32s srcStepB, Ipp32s mcTypeB, const
    Ipp16s *pSrcYData, Ipp32s srcYDataStep, Ipp8u *pDst, Ipp32s dstStep, Ipp32s
    roundControl);
```

### Parameters

<i>pSrcRefF</i>	Pointer to the forward reference block.
<i>srcStepF</i>	Size of the row in bytes, specifying the aligned forward reference frame width.
<i>mcTypeF</i>	Forward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcRefB</i>	Pointer to the backward reference block.
<i>srcStepB</i>	Size of row in bytes, specifying the aligned backward reference frame width.
<i>mcTypeB</i>	Backward MC type <a href="#">IPPVC MC APX</a> .
<i>pSrcYData</i>	Pointer to the data obtained after inverse DCT.
<i>srcYDataStep</i>	Number of bytes, specifying the width of the aligned data obtained after inverse DCT.
<i>pDst</i>	Pointer to the destination predicted block.
<i>dstStep</i>	Size of the row in bytes, specifying the aligned destination frame width.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiMC16x8BUV_8u_C1` calculates sum of 16x8 residual UV block and 16x8 predicted UV block, which is calculated as average of two 16x8 predictions (F-prediction and B-prediction) (See [Figure 16-2](#)). Each prediction is calculated on the basis of the corresponding reference block and corresponding *mcType* (see [IPPVC MC APX](#)).

Half sample prediction is calculated taking into account structure of [UV block](#). So (horizontally) neighboring elements of U-block (or V-block) have indexes  $i$  and  $i+2$  in UV block.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## Copy8x8, Copy16x16

*Copy fixed size block.*

---

### Syntax

```
IppStatus ippCopy8x8_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep);
IppStatus ippCopy16x16_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep);
```

### Parameters

<code>pSrc</code>	Pointer to the source block.
<code>srcStep</code>	Step in bytes through the source plane.
<code>pDst</code>	Pointer to the destination block.
<code>dstStep</code>	Step in bytes through the destination plane.

### Description

The functions `ippCopy8x8_8u_C1R` and `ippCopy16x16_8u_C1R` are declared in the `ippvc.h` file. These functions copy the source block to the destination block.

These functions are used in the H.261, H.263, and MPEG-4 encoders and decoders included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

`ippStsNullPtrErr` Indicates an error when at least one input pointer is NULL.

---

## Copy8x4HP, Copy8x8HP, Copy16x8HP, Copy16x16HP

*Copy fixed size blocks with half-pixel accuracy.*

---

### Syntax

```
IppStatus ippiCopy8x4HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
    dstStep, int acc, int rounding);
IppStatus ippiCopy8x8HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
    dstStep, int acc, int rounding);
IppStatus ippiCopy16x8HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, int acc, int rounding);
IppStatus ippiCopy16x16HP_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
    int dstStep, int acc, int rounding);
```

### Parameters

<i>pSrc</i>	Pointer to the source block.
<i>srcStep</i>	Step in bytes through the source plane.
<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Step in bytes through the destination plane.
<i>acc</i>	Parameter that determines half-pixel accuracy.
<i>rounding</i>	Parameter that determines type of rounding for pixel interpolation; may be 0 or 1.

### Description

The functions `ippiCopy8x4HP_8u_C1R`, `ippiCopy8x8HP_8u_C1R`, `ippiCopy16x8HP_8u_C1R`, and `ippiCopy16x16HP_8u_C1R` are declared in the `ippvc.h` file. These functions copy the source block to the destination block with half-pixel accuracy. Parameter *rounding* has the same meaning as `RTYPE` in [ITUH263] and `vop_rounding_type` in



[ISO14496]. Parameter *acc* is a two-bit value. Bit 0 defines half-pixel offset in horizontal direction and bit 1 defines half-pixel offset in vertical direction. The process of half-pixel interpolation is described in [ITUH263] subclause 6.1.2 and in [ISO14496] subclause 7.6.2.1.

The functions `ippiCopy8x4HP_8u_C1R` and `ippiCopy16x8HP_8u_C1R` are used in the MPEG-4 encoders and decoders included into IPP Samples. The functions `ippiCopy8x8HP_8u_C1R` and `ippiCopy16x16HP_8u_C1R` are used in the H.263 and MPEG-4 encoders and decoders included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippiStsNoErr</code>	Indicates no error.
<code>ippiStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## InterpolateAverage8x4, InterpolateAverage8x8, InterpolateAverage16x8, InterpolateAverage16x16

*Interpolate source block according to half-pixel offset and average the result with destination block.*

---

### Syntax

```
IppStatus ippiInterpolateAverage8x4_8u_C1R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pSrcDst, int srcDstStep, int acc, int rounding);
IppStatus ippiInterpolateAverage8x8_8u_C1R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pSrcDst, int srcDstStep, int acc, int rounding);
IppStatus ippiInterpolateAverage16x8_8u_C1R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pSrcDst, int srcDstStep, int acc, int rounding);
IppStatus ippiInterpolateAverage16x16_8u_C1R(const Ipp8u* pSrc, int srcStep,
      Ipp8u* pSrcDst, int srcDstStep, int acc, int rounding);
```

### Parameters

<i>pSrc</i>	Pointer to the source block.
<i>srcStep</i>	Step in bytes through the source plane.

<i>pSrcDst</i>	Pointer to the destination block.
<i>srcDstStep</i>	Step in bytes through the destination plane.
<i>acc</i>	Parameter that determines half-pixel offset.
<i>rounding</i>	Parameter that determines type of rounding for pixel interpolation; may be 0 or 1.

### Description

The functions `ippiInterpolateAverage8x4_8u_C1IR`, `ippiInterpolateAverage8x8_8u_C1IR`, `ippiInterpolateAverage16x8_8u_C1IR`, and `ippiInterpolateAverage16x16_8u_C1IR` are declared in the `ippvc.h` file. These functions interpolate the source block according to half-pixel offset and average the result with the destination block. Parameters *rounding* has the meaning as `RTYPE` in [ITUH263] and `vop_rounding_type` in [ISO14496]. Parameter *acc* is a two-bit value. Bit 0 defines half-pixel offset in horizontal direction and bit 1 defines half-pixel offset in vertical direction. These functions are used in MPEG-2 encoder and decoder included into IPP Samples together with functions [Copy8x4HP](#), [Copy8x8HP](#), [Copy16x8HP](#), [Copy16x16HP](#) for bidirectional motion compensation. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## Add8x8

*Adds two blocks with saturation.*

---

### Syntax

```
IppStatus ippiAdd8x8_16s8u_C1IRS(const Ipp16s* pSrc, int srcStep, Ipp8u*
    pSrcDst, int srcDstStep);
```

### Parameters

<i>pSrc</i>	Pointer to the source block.
<i>srcStep</i>	Step in bytes through the source plane.

---

<code>pSrcDst</code>	Pointer to the second source/destination block.
<code>srcDstStep</code>	Step in bytes through the destination plane.

### Description

The function `ippiAdd8x8_16s8u_C1IRS` is declared in the `ippvc.h` file. This function adds 16s data from `pSrc` block to the 8u data from `pSrcDst` block with saturation and stores the result in `pSrcDst` block. It can be used in motion compensation process to add a reconstructed block of prediction errors to the predictor.

This function is used in the H.261, H.263, and MPEG-4 encoders and decoders included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## Add8x8HP

*Adds blocks interpolated with half-pixel accuracy prediction to difference with saturation.*

---

### Syntax

```
IppStatus ippiAdd8x8HP_16s8u_C1RS(const Ipp16s* pSrc1, int src1Step, Ipp8u* pSrc2,
    int src2Step, Ipp8u* pDst, int dstStep, int acc, int rounding);
```

### Parameters

<code>pSrc1</code>	Pointer to the source block of differences.
<code>src1Step</code>	Step in bytes through the <code>Src1</code> plane.
<code>pSrc2</code>	Pointer to the source block of prediction.
<code>src2Step</code>	Step in bytes through the <code>Src2</code> plane.
<code>pDst</code>	Pointer to the second source/destination block.
<code>dstStep</code>	Step in bytes through the destination plane.
<code>acc</code>	Parameter that determines half-pixel accuracy.

<i>rounding</i>	Parameter that determines type of rounding for pixel interpolation; may be 0 or 1.
-----------------	--

### Description

The function `ippiAdd8x8HP_16s8u_C1RS` is declared in the `ippvc.h` file. This function adds 16s data from *pSrc1* block to the data from *pSrc2* block interpolated with half-pixel accuracy with saturation. It can be used in motion compensation process to add a reconstructed block of prediction errors to the predictor. Parameter *rounding* has the same meaning as RTYPE in [ITUH263] and `vop_rounding_type` in [ISO14496]. Parameter *acc* is a two-bit value. Bit 0 defines half-pixel offset in horizontal direction and bit 1 defines half-pixel offset in vertical direction. The process of half-pixel interpolation is described in [ITUH263] subclause 6.1.2 and in [ISO14496] subclause 7.6.2.1.

This function is used in the H.263 and MPEG-4 encoders and decoders included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## AddC8x8

*Adds a constant to 8x8 block with saturation.*

---

### Syntax

```
IppStatus ippiAddC8x8_16s8u_C1IR(Ipp16s value, Ipp8u* pSrcDst, int srcDstStep);
```

### Parameters

<i>value</i>	Constant value to be added to the block.
<i>pSrcDst</i>	Pointer to the source/destination block.
<i>srcDstStep</i>	Step in bytes through the destination block.

### Description

The function `ippiAddC8x8_16s8u_C1IR` is declared in the `ippvc.h` file. This function adds a 16s value to an 8u block of 8x8 size with saturation.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## Average8x8, Average16x16

*Average two blocks.*

---

## Syntax

```
IppStatus ippAverage8x8_8u_C1IR(Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst, int  
    srcDstStep);  
IppStatus ippAverage16x16_8u_C1IR(Ipp8u* pSrc, int srcStep, Ipp8u* pSrcDst,  
    int srcDstStep);
```

## Parameters

<code>pSrc, pSrcDst</code>	Pointers to the source/destination blocks.
<code>srcStep, srcDstStep</code>	Width in bytes through the source/destination planes.

## Description

The functions `ippAverage8x8_8u_C1IR` and `ippAverage16x16_8u_C1IR` are declared in the `ippvc.h` file. These functions average two blocks pixel-by-pixel, as specified by [\[ISO14496\]](#), subclause 7.6.9.4. The functions can be used in the B-frame reconstruction.

These functions are used in the MPEG-4 encoders and decoders included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

Motion Estimation

These functions calculate:

- residual block - difference between source block and predicted block ([Table 16-8](#))
- some characteristics of the residual block ([Table 16-9](#))
- some characteristics of the blocks. These characteristics can be used for comparison of the blocks. ([Table 16-10](#)).

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm> .

Table 16-8 Evaluation of Residual Block

<i>For predicted blocks</i>	<i>For bi-predicted blocks</i>
<a href="#">GetDiff16x16</a>	<a href="#">GetDiff16x16B</a>
<a href="#">GetDiff16x8</a>	<a href="#">GetDiff16x8B</a>
<a href="#">GetDiff8x8</a>	<a href="#">GetDiff8x8B</a>
<a href="#">GetDiff8x16</a>	<a href="#">GetDiff8x16B</a>
<a href="#">GetDiff8x4</a>	<a href="#">GetDiff8x4B</a>
<a href="#">GetDiff4x4</a>	
<a href="#">Sub8x8, Sub16x16</a>	
<a href="#">SubSAD8x8</a>	

Table 16-9 Evaluation of Residual Block Characteristics

<i>For predicted blocks</i>	<i>For bi-predicted blocks</i>
<a href="#">SqrDiff16x16</a>	<a href="#">SqrDiff16x16B</a>
<a href="#">SSD8x8</a>	
<a href="#">SSD4x4</a>	
<b>Evaluation of variances and means of blocks of difference between two blocks</b>	
<a href="#">VarMeanDiff16x16</a>	
<a href="#">VarMeanDiff16x8</a>	
<b>SAD functions</b>	
<a href="#">SAD16x16</a>	

Table 16-9      Evaluation of Residual Block Characteristics (continued)

<a href="#">SAD16x8</a>
<a href="#">SAD8x8</a>
<a href="#">SAD4x4</a>
<a href="#">SAD16x16Blocks8x8</a>
<a href="#">SAD16x16Blocks4x4</a>
<a href="#">FrameFieldSAD16x16</a>
<b>Sum of differences evaluation</b>
<a href="#">SumsDiff16x16Blocks4x4</a>
<a href="#">SumsDiff8x8Blocks4x4</a>

Table 16-10      Evaluation of Blocks Characteristics

<b>Block variance and mean evaluation</b>
<a href="#">VarMean8x8</a>
<b>Block variance evaluation</b>
<a href="#">Variance16x16</a>
<b>Evaluation of block deviation</b>
<a href="#">MeanAbsDev16x16</a>
<b>Edges detection</b>
<a href="#">EdgesDetect16x16</a>

Difference Evaluation

These functions calculate residual block - difference between the source block and the predicted block.

Difference evaluation functions are divided into two groups:

- Functions for predicted block, using one reference block for prediction
- Functions for bi-predicted block, using two reference blocks for prediction; in this case predictions are first calculated for each reference block, and then prediction is calculated as average of two predictions.

## Predicted Blocks

These functions use one reference block for prediction.

---

## GetDiff16x16

*Evaluates difference between current predicted and reference blocks of 16x16 elements.*

---

### Syntax

```
IppStatus ippiGetDiff16x16_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s mcType,
Ipp32s roundControl);
```

### Parameters

<i>pSrcCur</i>	Pointer to the block of size 16x16 in the current plane.
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>pSrcRef</i>	Pointer to the block of size 16x16 in the reference plane.
<i>srcRefStep</i>	Step of the reference block, specifying width of the plane in bytes.
<i>pDstDiff</i>	Pointer to the block of size 16x16 in the destination plane, which contains difference between the current and reference blocks.
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>pDstPredictor</i>	Pointer to the destination block of size 16x16 that contains a block of predictors for the current block. Predictor is calculated from the reference block taking into account <i>mcType</i> . If predictor is not used, it must be 0.
<i>dstPredictorStep</i>	Step of the <i>pDstPredictor</i> block in bytes.
<i>mcType</i>	Type of the following MC type <a href="#">IPPVC MC APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.



## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff16x16_8u16s_C1` evaluates the difference between the current block of specified size and the reference one. The result is stored in blocks `pDstDiff` and `pDstPredictor`. The latter stores some additional information about the coding block. This information is used for encoding next blocks that refer to the current one. This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## GetDiff16x8

*Evaluates difference between current predicted and reference blocks of 16x8 elements.*

---

## Syntax

```
IppStatus ippiGetDiff16x8_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s mcType,
    Ipp32s roundControl);
```

## Parameters

<code>pSrcCur</code>	Pointer to the block of size 16x8 in the current plane.
<code>srcCurStep</code>	Step of the current block, specifying width of the plane in bytes.
<code>pSrcRef</code>	Pointer to the block of size 16x8 in the reference plane.
<code>srcRefStep</code>	Step of the reference block, specifying width of the plane in bytes.
<code>pDstDiff</code>	Pointer to the block of size 16x8 in the destination plane, which contains difference between the current and reference blocks.
<code>dstDiffStep</code>	Step of the destination block, specifying width of the block in bytes.

<i>pDstPredictor</i>	Pointer to the destination block of size 16x8 that contains a block of predictors for the current block. Predictor is calculated from the reference block taking into account <i>mcType</i> . If predictor is not used, it must be 0.
<i>dstPredictorStep</i>	Step of the <i>pDstPredictor</i> block in bytes.
<i>mcType</i>	Type of the following MC type <a href="#">IPPVC MC APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff16x8_8u16s_C1` evaluates the difference between the current block of specified size and the reference one. The result is stored in blocks *pDstDiff* and *pDstPredictor*. The latter stores some additional information about the coding block. This information is used for encoding next blocks that refer to the current one. This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## GetDiff8x8

*Evaluates difference between current predicted and reference blocks of 8x8 elements.*

---

## Syntax

```
IppStatus ippiGetDiff8x8_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s mcType,
Ipp32s roundControl);
```

## Parameters

<i>pSrcCur</i>	Pointer to the block of size 8x8 in the current plane.
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.

---

<i>pSrcRef</i>	Pointer to the block of size 8x8 in the reference plane.
<i>srcRefStep</i>	Step of the reference block, specifying width of the plane in bytes.
<i>pDstDiff</i>	Pointer to the block of size 8x8 in the destination plane, which contains difference between the current and reference blocks.
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>pDstPredictor</i>	Pointer to the destination block of size 8x8 that contains a block of predictors for the current block. Predictor is calculated from the reference block taking into account <i>mcType</i> . If predictor is not used, it must be 0.
<i>dstPredictorStep</i>	Step of the <i>pDstPredictor</i> block in bytes.
<i>mcType</i>	Type of the following MC type <a href="#">IPPVC MC APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x8_8u16s_C1` evaluates the difference between the current block of specified size and the reference one. The result is stored in blocks *pDstDiff* and *pDstPredictor*. The latter stores some additional information about the coding block. This information is used for encoding next blocks that refer to the current one. This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## GetDiff8x16

*Evaluates difference between current predicted and reference blocks of 8x16 elements.*

---

### Syntax

```
IppStatus ippGetDiff8x16_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,  
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s  
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s mcType,  
    Ipp32s roundControl);
```

### Parameters

<i>pSrcCur</i>	Pointer to the block of size 8x16 in the current plane.
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>pSrcRef</i>	Pointer to the block of size 8x16 in the reference plane.
<i>srcRefStep</i>	Step of the reference block, specifying width of the plane in bytes.
<i>pDstDiff</i>	Pointer to the block of size 8x16 in the destination plane, which contains difference between the current and reference blocks.
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>pDstPredictor</i>	Pointer to the destination block of size 8x16 that contains a block of predictors for the current block. Predictor is calculated from the reference block taking into account <i>mcType</i> . If predictor is not used, it must be 0.
<i>dstPredictorStep</i>	Step of the <i>pDstPredictor</i> block in bytes.
<i>mcType</i>	Type of the following MC type <a href="#">IPPVC MC APX</a> .
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippGetDiff8x16_8u16s_C1` evaluates the difference between the current block of specified size and the reference one. The result is stored in blocks *pDstDiff* and *pDstPredictor*. The

latter stores some additional information about the coding block. This information is used for encoding next blocks that refer to the current one. This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## GetDiff8x4

*Evaluates difference between current predicted and reference blocks of 8x4 elements.*

---

### Syntax

```
IppStatus ippGetDiff8x4_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s mcType,
    Ipp32s roundControl);
```

### Parameters

<code>pSrcCur</code>	Pointer to the block of size 8x4 in the current plane.
<code>srcCurStep</code>	Step of the current block, specifying width of the plane in bytes.
<code>pSrcRef</code>	Pointer to the block of size 8x4 in the reference plane.
<code>srcRefStep</code>	Step of the reference block, specifying width of the plane in bytes.
<code>pDstDiff</code>	Pointer to the block of size 8x4 in the destination plane, which contains difference between the current and reference blocks.
<code>dstDiffStep</code>	Step of the destination block, specifying width of the block in bytes.
<code>pDstPredictor</code>	Pointer to the destination block of size 8x4 that contains a block of predictors for the current block. Predictor is calculated from the reference block taking into account <code>mcType</code> . If predictor is not used, it must be 0.
<code>dstPredictorStep</code>	Step of the <code>pDstPredictor</code> block in bytes.
<code>mcType</code>	Type of the following MC type <a href="#">IPPVC MC APX</a> .

*roundControl*     Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x4_8u16s_C1` evaluates the difference between the current block of specified size and the reference one. The result is stored in blocks *pDstDiff* and *pDstPredictor*. The latter stores some additional information about the coding block. This information is used for encoding next blocks that refer to the current one. This method helps to decrease the number of encoding errors. Encoding is performed accurate to half a pel and rounding must be specified.

### Return Values

`ippStsNoErr`             Indicates no error.

`ippStsNullPtrErr`     Indicates an error when at least one input pointer is NULL.

---

## GetDiff4x4

*Computes difference between current predicted and reference 4x4 blocks.*

---

### Syntax

```
IppStatus ippiGetDiff4x4_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRef, Ipp32s srcRefStep, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp16s* pDstPredictor, Ipp32s dstPredictorStep, Ipp32s mcType,
    Ipp32s roundControl);
```

### Parameters

*pSrcCur*             Pointer to the block of size 4x4 in the current plane.

*srcCurStep*         Step of the current block, specifying width of the plane in bytes.

*pSrcRef*             Pointer to the block of size 4x4 in the reference plane.

*srcRefStep*         Step of the reference block, specifying width of the plane in bytes.

*pDstDiff*            Pointer to the block of size 4x4 in the destination plane, which contains difference between the current and reference blocks.

*dstDiffStep*         Step of the destination block, specifying width of the block in bytes.

`pDstPredictor` Reserved parameter (must be 0).  
`dstPredictorStep` Reserved parameter (must be 0).  
`mcType` Reserved parameter (must be 0).  
`roundControl` Reserved parameter (must be 0).

## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff4x4_8u16s_C1` computes the difference between predictor 4x4 block and source 4x4 block.

## Return Values

`ippStsNoErr` Indicates no error.  
`ippStsNullPtrErr` Indicates an error when at least one input pointer is NULL.

## Bi-Predicted Blocks

These functions use two reference blocks for prediction. At first, predictions are calculated for each reference block, and then prediction is calculated as average of two predictions.

---

## GetDiff16x16B

*Evaluates difference between current bi-predicted and mean of two reference blocks of 16x16 elements.*

---

## Syntax

```
IppStatus ippiGetDiff16x16B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
dstDiffStep, Ipp32s roundControl);
```

## Parameters

`pSrcCur` Pointer to the block of size 16x16 in the current plane.  
`srcCurStep` Step of the current block, specifying width of the plane in bytes.

<i>pSrcRefF</i>	Pointer to the forward block of size 16x16 in the reference plane.
<i>srcRefStepF</i>	Step of the forward reference block, specifying width of the plane in bytes.
<i>mcTypeF</i>	Type of the following forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward block of size 16x16 in the reference plane.
<i>srcRefStepB</i>	Step of the backward reference block, specifying width of the block in bytes.
<i>mcTypeB</i>	Type of the following backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pDstDiff</i>	Pointer to the destination block of size 16x16 that contains difference between current and reference blocks.
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff16x16B_8u16s_C1` evaluates the difference between the current block and the mean of two reference blocks of specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff*. Encoding is performed accurate to half a pel and rounding must be specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.



## GetDiff16x8B

*Evaluates difference between current bi-predicted and mean of two reference blocks of 16x8 elements.*

### Syntax

```
IppStatus ippiGetDiff16x8B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
    pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp32s roundControl);
```

### Parameters

<i>pSrcCur</i>	Pointer to the block of size 16x8 in the current plane.
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>pSrcRefF</i>	Pointer to the forward block of size 16x8 in the reference plane.
<i>srcRefStepF</i>	Step of the forward reference block, specifying width of the plane in bytes.
<i>mcTypeF</i>	Type of the following forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward block of size 16x8 in the reference plane.
<i>srcRefStepB</i>	Step of the backward reference block, specifying width of the block in bytes.
<i>mcTypeB</i>	Type of the following backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pDstDiff</i>	Pointer to the destination block of specified size containing difference between the current and reference blocks.
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff16x8B_8u16s_C1` evaluates the difference between the current block and the mean of two reference blocks of specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff*. Encoding is performed accurate to half a pel and rounding must be specified.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## GetDiff8x8B

*Evaluates difference between current bi-predicted and mean of two reference blocks of 8x8 elements.*

---

### Syntax

```
IppStatus ippGetDiff8x8B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
    pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp32s roundControl);
```

### Parameters

<code>pSrcCur</code>	Pointer to the block of size 8x8 in the current plane.
<code>srcCurStep</code>	Step of the current block, specifying width of the plane in bytes.
<code>pSrcRefF</code>	Pointer to the forward block of size 8x8 in the reference plane.
<code>srcRefStepF</code>	Step of the forward reference block, specifying width of the plane in bytes.
<code>mcTypeF</code>	Type of the following forward MC type <a href="#">IPPVC MC APX</a> .
<code>pSrcRefB</code>	Pointer to the backward block of size 8x8 in the reference plane.
<code>srcRefStepB</code>	Step of the backward reference block, specifying width of the block in bytes.
<code>mcTypeB</code>	Type of the following backward MC type <a href="#">IPPVC MC APX</a> .
<code>pDstDiff</code>	Pointer to the destination block of specified size containing difference between the current and reference blocks.
<code>dstDiffStep</code>	Step of the destination block, specifying width of the block in bytes.
<code>roundControl</code>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x8B_8u16s_C1` evaluates the difference between the current block and the mean of two reference blocks of specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block `pDstDiff`. Encoding is performed accurate to half a pel and rounding must be specified.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## GetDiff8x16B

*Evaluates difference between current bi-predicted and mean of two reference blocks of 8x16 elements.*

---

## Syntax

```
IppStatus ippiGetDiff8x16B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
    const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
    pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
    dstDiffStep, Ipp32s roundControl);
```

## Parameters

<code>pSrcCur</code>	Pointer to the block of size 8x16 in the current plane.
<code>srcCurStep</code>	Step of the current block, specifying width of the plane in bytes.
<code>pSrcRefF</code>	Pointer to the forward block of size 8x16 in the reference plane.
<code>srcRefStepF</code>	Step of the forward reference block, specifying width of the plane in bytes.
<code>mcTypeF</code>	Type of the following forward MC type <a href="#">IPPVC MC APX</a> .
<code>pSrcRefB</code>	Pointer to the backward block of size 8x16 in the reference plane.
<code>srcRefStepB</code>	Step of the backward reference block, specifying width of the block in bytes.
<code>mcTypeB</code>	Type of the following backward MC type <a href="#">IPPVC MC APX</a> .

<i>pDstDiff</i>	Pointer to the destination block of specified size containing difference between the current and reference blocks.
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x16B_8u16s_C1` evaluates the difference between the current block and the mean of two reference blocks of specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff*. Encoding is performed accurate to half a pel and rounding must be specified.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## GetDiff8x4B

*Evaluates difference between current bi-predicted and mean of two reference blocks of 8x4 elements.*

---

### Syntax

```
IppStatus ippiGetDiff8x4B_8u16s_C1(const Ipp8u* pSrcCur, Ipp32s srcCurStep,
const Ipp8u* pSrcRefF, Ipp32s srcRefStepF, Ipp32s mcTypeF, const Ipp8u*
pSrcRefB, Ipp32s srcRefStepB, Ipp32s mcTypeB, Ipp16s* pDstDiff, Ipp32s
dstDiffStep, Ipp32s roundControl);
```

### Parameters

<i>pSrcCur</i>	Pointer to the block of size 8x4 in the current plane.
<i>srcCurStep</i>	Step of the current block, specifying width of the plane in bytes.
<i>pSrcRefF</i>	Pointer to the forward block of size 8x4 in the reference plane.
<i>srcRefStepF</i>	Step of the forward reference block, specifying width of the plane in bytes.

---

<i>mcTypeF</i>	Type of the following forward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pSrcRefB</i>	Pointer to the backward block of size 8x4 in the reference plane.
<i>srcRefStepB</i>	Step of the backward reference block, specifying width of the block in bytes.
<i>mcTypeB</i>	Type of the following backward MC type <a href="#">IPPVC_MC_APX</a> .
<i>pDstDiff</i>	Pointer to the destination block of specified size containing difference between the current and reference blocks.
<i>dstDiffStep</i>	Step of the destination block, specifying width of the block in bytes.
<i>roundControl</i>	Parameter that determines type of rounding for half a pel approximation; may be 0 or 1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiGetDiff8x4B_8u16s_C1` evaluates the difference between the current block and the mean of two reference blocks of specified size. One of the reference blocks is called forward and belongs to the previous frame in accordance with the type of motion compensation. The other block is called backward and belongs to one of the following frames. The result is stored in block *pDstDiff*. Encoding is performed accurate to half a pel and rounding must be specified.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## Sub8x8, Sub16x16

*Subtract two blocks and store the result in the third block.*

---

### Syntax

```
IppStatus ippiSub8x8_8u16s_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp16s* pDst, int dstStep);

IppStatus ippiSub16x16_8u16s_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u*
    pSrc2, int src2Step, Ipp16s* pDst, int dstStep);
```

## Parameters

<i>pSrc1</i>	Pointer to the first source block.
<i>src1Step</i>	Step in bytes through the first source plane.
<i>pSrc2</i>	Pointer to the second source block.
<i>src2Step</i>	Step in bytes through the second source plane.
<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Step in bytes through the destination plane.

## Description

The functions `ippiSub8x8_8u16s_C1R` and `ippiSub16x16_8u16s_C1R` are declared in the `ippvc.h` file. These functions subtract two blocks and store the result in *pDst* block. The functions can be used in motion estimation process to get a difference of current and reference blocks.

These functions are used in the H.261, H.263, and MPEG-4 encoders included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## SubSAD8x8

*Subtracts two blocks, stores the result in the third block and computes a sum of absolute differences.*

---

## Syntax

```
IppStatus ippiSubSAD8x8_8u16s_C1R(const Ipp8u* pSrc1, int src1Step, const
    Ipp8u* pSrc2, int src2Step, Ipp16s* pDst, int dstStep, Ipp32s* pSAD);
```

## Parameters

<i>pSrc1</i>	Pointer to the first source block.
<i>src1Step</i>	Step in bytes through the first source plane.

<i>pSrc2</i>	Pointer to the second source block.
<i>src2Step</i>	Step in bytes through the second source plane.
<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Step in bytes through the destination plane.
<i>pSAD</i>	Pointer to the resulting SAD.

## Description

The function `ippiSubSAD8x8_8u16s_C1R` is declared in the `ippvc.h` file. This function subtracts two blocks and stores the result in *pDst* block and stores SAD of the result in *pSAD*. This function can be used in motion estimation process to get a difference of current and reference blocks.

This function is used in the H.261, H.263, and MPEG-4 encoders included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## Sum of Squares of Differences Evaluation

These functions evaluate sum of squares of differences between current and predicted blocks.

These functions are divided into two groups:

- Functions for predicted block, using one reference block for prediction
- Functions for bi-predicted block, using two reference blocks for prediction; in this case predictions are first calculated for each reference block, and then prediction is calculated as average of two predictions.

## SqrDiff16x16

*Evaluates sum of squares of differences between current and reference 16X16 blocks.*

---

### Syntax

```
IppStatus ippISqrDiff16x16_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, const
    Ipp8u* pRef, Ipp32s refStep, Ipp32s mcType, Ipp32s* pSqrDiff);
```

### Parameters

<i>pSrc</i>	Pointer to the current block of specified size.
<i>srcStep</i>	Step of the current block, specifying width of the block in bytes.
<i>pRef</i>	Pointer to the reference block of specified size.
<i>refStep</i>	Step of the reference block, specifying width of the block in bytes.
<i>mcType</i>	MC type <a href="#">IPPVC MC APX</a> .
<i>pSqrDiff</i>	Pointer to the sum of square difference between all elements in the current and reference blocks.

### Description

This function is declared in the `ippvc.h` header file. The function `ippISqrDiff16x16_8u32s` evaluates the sum of square difference between all the elements in the current block and corresponding elements in the reference block. The result is stored in integer *\*pSqrDiff*.

Let us denote the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the current block as  $block[i, j]$  and the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the reference block as  $ref\_block[i, j]$ . Then

$$*pSqrDiff = \sum_{i=0}^{15} \sum_{j=0}^{15} (block[i, j] - ref\_block[i, j])^2.$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.



---

`ippStsStepErr` Indicates an error when `srcStep` or `refStep` is less or equal to zero.

---

## SqrDiff16x16B

*Evaluates sum of squares of differences between current bi-predicted 16X16 block and mean of two reference blocks.*

---

### Syntax

```
IppStatus ippiSqrDiff16x16B_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, const
    Ipp8u pRefF, Ipp32s refStepF, Ipp32s mcTypeF, const Ipp8u pRefB, Ipp32s
    refStepB, Ipp32s mcTypeB, Ipp32s* pSqrDiff);
```

### Parameters

<code>pSrc</code>	Pointer to the current block of specified size.
<code>srcStep</code>	Step of the current block, specifying width of the block in bytes.
<code>pRefF</code>	Pointer to the forward reference block of specified size.
<code>refStepF</code>	Step of the forward reference block, specifying width of the block in bytes.
<code>mcTypeF</code>	MC type <a href="#">IPPVC MC APX</a> for the forward reference block.
<code>pRefB</code>	Pointer to the backward reference block of specified size.
<code>refStepB</code>	Step of the backward reference block, specifying width of the block in bytes.
<code>mcTypeB</code>	MC type <a href="#">IPPVC MC APX</a> for the backward reference block.
<code>pSqrDiff</code>	Pointer to the sum of square difference between all elements in the current and reference blocks.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiSqrDiff16x16B_8u32s` evaluates the sum of square difference between all the elements in the current block and the mean of corresponding elements in two reference blocks of 8x8 elements. The result is stored in integer `pSqrDiff`.

Let us denote the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the current block as  $block[i, j]$ , the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the forward reference block as  $f\_block[i, j]$ , and the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the forward reference block as  $b\_block[i, j]$ . Then

$$*pSqrDiff = \sum_{i=0}^{15} \sum_{j=0}^{15} \left( block[i, j] - \frac{f\_block[i, j] + b\_block[i, j]}{2} \right)^2$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> , <i>refStepF</i> or <i>refStepB</i> is less or equal to zero.

---

## SSD8x8

*Evaluates sum of squares of differences between current and reference 8X8 blocks.*

---

### Syntax

```
IppStatus ippISSD8x8_8u32s_C1R(const Ipp8u *srcStep, int srcCurStep, const
    Ipp8u* pSrcRef, int srcRefStep, Ipp32s *pDst, Ipp32s mcType);
```

### Parameters

<i>pSrc</i>	Pointer to the current block of specified size.
<i>srcCurStep</i>	Step of the current block, specifying width of the block in bytes.
<i>pSrcRef</i>	Pointer to the reference block of specified size.
<i>srcRefStep</i>	Step of the reference block, specifying width of the block in bytes.
<i>pDst</i>	Pointer to the sum of square difference between all elements in the current and reference blocks.
<i>mcType</i>	MC type <a href="#">IPPVC</a> <a href="#">MC</a> <a href="#">APX</a> .

## Description

This function is declared in the `ippvc.h` header file. The function `ippiSSD8x8_8u32s_C1R` evaluates the sum of square difference between all the elements in the current block and corresponding elements in the reference block. The result is stored in integer `*pDst`.

Let us denote the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the current block as  $block[i, j]$  and the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the reference block as  $ref\_block[i, j]$ . Then

$$*pSqrDiff = \sum_{i=0}^{15} \sum_{j=0}^{15} (block[i, j] - ref\_block[i, j])^2.$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## SSD4x4

*Evaluates sum of squares of differences between current and reference 4X4 blocks.*

---

## Syntax

```
IppStatus ippiSSD4x4_8u32s_C1R(const Ipp8u *srcStep, int srcCurStep, const
    Ipp8u* pSrcRef, int srcRefStep, Ipp32s *pDst, Ipp32s mcType);
```

## Parameters

<code>pSrc</code>	Pointer to the current block of specified size.
<code>srcCurStep</code>	Step of the current block, specifying width of the block in bytes.
<code>pSrcRef</code>	Pointer to the reference block of specified size.
<code>srcRefStep</code>	Step of the reference block, specifying width of the block in bytes.
<code>pDst</code>	Pointer to the sum of square difference between all elements in the current and reference blocks.

`mcType`                      MC type [IPPVC\\_MC\\_APX](#).

### Description

This function is declared in the `ippvc.h` header file. The function `ippiSSD4x4_8u32s_C1R` evaluates the sum of square difference between all the elements in the current block and corresponding elements in the reference block. The result is stored in integer `*pDst`.

Let us denote the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the current block as `block[i, j]` and the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the reference block as `ref_block[i, j]`. Then

$$*pSqrDiff = \sum_{i=0}^{15} \sum_{j=0}^{15} (\text{block}[i, j] - \text{ref\_block}[i, j])^2.$$

### Return Values

`ippStsNoErr`                      Indicates no error.

`ippStsNullPtrErr`              Indicates an error when at least one input pointer is NULL.

### Block Variance and Mean Evaluation

These functions evaluate sum and variance of all elements in block.

---

## VarMean8x8

*Evaluates variance and mean of 8X8 block.*

---

### Syntax

```

IppStatus ippiVarMean8x8_8u32s_C1R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp32s*
    pVar, Ipp32s* pMean);
IppStatus ippiVarMean8x8_16s32s_C1R(const Ipp16s* pSrc, Ipp32s srcStep, Ipp32s*
    pVar, Ipp32s* pMean);

```

## Parameters

<i>pSrc</i>	Pointer to the input block 8x8.
<i>srcStep</i>	Input block step.
<i>pVar</i>	Pointer to the variance.
<i>pMean</i>	Pointer to the mean value.

## Description

This function is declared in the `ippvc.h` header file. The function evaluates the mean and variance of an 8x8 block of unsigned char values (`ippiVarMean8x8_8u32s_C1R`) or short integer values (`ippiVarMean8x8_16s32s_C1R`).

Mean is evaluated by formula:

$$M = \frac{\sum_{i=0}^7 \sum_{j=0}^7 block[i, j]}{64}$$

Variance is evaluated by formula:

$$V = \frac{\left( \sum_{i=0}^7 \sum_{j=0}^7 block[i, j]^2 \right) - M^2}{64}.$$

This function is used in the MPEG-2 encoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## Evaluation of Variances and Means of Blocks of Difference Between Two Blocks

These functions evaluate sum and variance of all elements in block.

---

### VarMeanDiff16x16

*Evaluates variances and means of four 8x8 blocks of difference between two 16x16 blocks.*

---

#### Syntax

```
IppStatus ippiVarMeanDiff16x16_8u32s_C1R(const Ipp8u* pSrc, Ipp32s srcStep,
    const Ipp8u* pRef, Ipp32s srcStep, Ipp32s* pSrcSum, Ipp32s* pVar, Ipp32s*
    pMean, Ipp32s mcType);
```

#### Parameters

<i>pSrc</i>	Pointer to the current block 16x16.
<i>srcStep</i>	Current block step.
<i>pRef</i>	Pointer to the reference block 16x16.
<i>srcStep</i>	Reference block step.
<i>pSrcSum</i>	Pointer to the sum of pixel values for the current block.
<i>pVar</i>	Pointer to the variance.
<i>pMean</i>	Pointer to the mean value.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .

#### Description

This function is declared in the `ippvc.h` header file. The function `ippiVarMeanDiff16x16_8u32s_C1R` evaluates the means and variances of four 8x8 blocks of difference between two 16x16 blocks.

As the function is repeatedly called for the same current block, the parameter *pSrcSum* is used to avoid excessive computation. This parameter should point to an array of four 8x8 blocks comprising the current block. The parameters *pVar*, *pMean* should point to arrays of four elements, since the values of variance and mean are calculated separately for each 8x8 block.

Mean is evaluated by formula:

$$M = \frac{\text{Sum} - \sum_{i=0}^7 \sum_{j=0}^7 \text{ref\_block}[i, j]}{64},$$

where *Sum* is the sum of pixel values for the current block taken from *pSrcSum* parameter.

Variance is evaluated by formula:

$$V = \frac{\left( \sum_{i=0}^7 \sum_{j=0}^7 (\text{cur\_block}[i, j] - \text{ref\_block}[i, j])^2 \right) - M^2}{64}.$$

This function is used in the MPEG-2 encoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## VarMeanDiff16x8

*Evaluates variances and means of two 8x8 blocks of difference between two 16x8 blocks.*

---

### Syntax

```
IppStatus ippiVarMeanDiff16x8_8u32s_C1R(const Ipp8u* pSrc, Ipp32s srcStep,
const Ipp8u* pRef, Ipp32s srcStep, Ipp32s* pSrcSum, Ipp32s* pVar, Ipp32s*
pMean, Ipp32s mcType);
```

### Parameters

<i>pSrc</i>	Pointer to the current block 16x8.
<i>srcStep</i>	Current block step.
<i>pRef</i>	Pointer to the reference block 16x8.
<i>srcStep</i>	Reference block step.

<i>pSrcSum</i>	Pointer to the sum of pixel values for the current block.
<i>pVar</i>	Pointer to the variance.
<i>pMean</i>	Pointer to the mean value.
<i>mcType</i>	MC type <a href="#">IPPVC_MC_APX</a> .

## Description

This function is declared in the `ippvc.h` header file. The function `ippiVarMeanDiff16x8_8u32s_C1R` evaluates the means and variances of two 8x8 blocks of difference between two 16x8 blocks.

As the function is repeatedly called for the same current block, the parameter *pSrcSum* is used to avoid excessive computation. This parameter should point to an array of four 8x8 blocks comprising the current block. The parameters *pVar*, *pMean* should point to arrays of four elements, since the values of variance and mean are calculated separately for each 8x8 block.

Mean is evaluated by formula:

$$M = \frac{\sum_{i=0}^7 \sum_{j=0}^7 ref\_block[i, j]}{64}$$

Variance is evaluated by formula:

$$V = \frac{\left( \sum_{i=0}^7 \sum_{j=0}^7 (cur\_block[i, j] - ref\_block[i, j])^2 \right) - M^2}{64}$$

This function is used in the MPEG-2 encoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.



## Block Variance Evaluation

This function evaluates variance of all elements in block.

---

## Variance16x16

*Evaluates variance of current block.*

---

### Syntax

```
IppStatus ippiVariance16x16_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, Ipp32s* pVar);
```

### Parameters

<i>pSrc</i>	Pointer to the current block of specified size.
<i>srcStep</i>	Step of the current block, specifying width of the block in bytes.
<i>pVar</i>	Pointer to block variance.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiVariance16x16_8u32s` evaluates variance of all the elements in the current block. The result is stored in integer `Var`.

Let us denote the pel which lies at the crossing of  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the current block as  $block[i, j]$ . Then

$$Var = \sum_{i=0}^{15} \sum_{j=0}^{15} block[i, j]^2 - \frac{\left( \sum_{i=0}^{15} \sum_{j=0}^{15} block[i, j] \right)^2}{256} .$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error when <i>srcStep</i> is less or equal to zero.

## Evaluation of Block Deviation

This function evaluates mean absolute deviation of a block.

---

## MeanAbsDev16x16

*Evaluates mean absolute deviation for a 16x16 block.*

---

### Syntax

```
IppStatus ippMeanAbsDev16x16_8u32s_C1R(const Ipp8u *pSrc, int srcStep, Ipp32s
    *pDst);
```

### Parameters

<i>pSrcCur</i>	Pointer to the block 16x16.
<i>srcStep</i>	Step of the block.
<i>pDst</i>	Pointer to the deviation.

### Description

The function `ippMeanAbsDev16x16_8u32s_C1R` is declared in the `ippvc.h` file. This function evaluates the mean absolute deviation for a 16x16 block by formula:

$$Dev = \sum_{i=0}^{15} \sum_{j=0}^{15} |block[i, j] - Mean|,$$

where *Mean* is evaluated by formula

$$Mean = \frac{\sum_{i=0}^{15} \sum_{j=0}^{15} block[i, j] + 128}{256}.$$

This function is used in the H.261, H.263, and MPEG-4 encoders included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## Edges Detection

This function detects edges inside block.

---

## EdgesDetect16x16

*Detects edges inside 16X16 block.*

---

### Syntax

```
IppStatus ippiEdgesDetect16x16_8u_C1R(const Ipp8u *pSrc, Ipp32u srcStep, Ipp8u  
EdgePelDifference, Ipp8u EdgePelCount, Ipp8u *pRes);
```

### Parameters

<code>pSrc</code>	Pointer to a 16x16 block in the current plan.
<code>srcStep</code>	Step of the current block, specifying width of the plane in bytes.
<code>EdgePelDifference</code>	The value for estimation of difference between neighboring elements. This value must be within the range of [0, 128].
<code>EdgePelCount</code>	The value for estimation of number of pairs of elements with ‘big difference’. This value must be within the range of [0, 128].
<code>pRes</code>	Pointer to output value. ( <code>*pRes</code> ) is equal to 1, if edges are detected, and is equal to 0 if edges are not detected.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiEdgesDetect16x16_8u_C1R` detects edges inside a 16x16 block: finds pair of neighboring (horizontal and vertical) elements with difference greater than `EdgePelDifference`.

If the number of pairs is greater than *EdgePelCount*, edges are detected and flag (*\*pRes*) is set to 1. Otherwise, edges are not detected (*\*pRes*) is set to 0).

```
res = 0  count = 0  row ∈ [0,14]  col ∈ [0,14]

if(Src[row][col] - Src[row][col + 1]) > EdgePelDifference → count++

if(Src[row][col] - Src[row + 1][col]) > EdgePelDifference → count++

if(count > EdgePelCount) → res = 1.
```

This function is used in the H.264 encoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error when at least one input pointer is NULL.

### SAD Functions

These functions evaluate sum of absolute difference between current and predicted blocks.

---

## SAD16x16

*Evaluates sum of absolute difference between current and reference 16X16 blocks.*

---

### Syntax

```
IppStatus ippisAD16x16_8u32s(const Ipp8u* pSrc, Ipp32s srcStep, const Ipp8u*
    pRef, Ipp32s refStep, Ipp32s* pSAD, Ipp32s mcType);
```

### Parameters

<i>pSrc</i>	Pointer to the current block of specified size.
<i>srcStep</i>	Step of the current block, specifying width of the block in bytes.
<i>pRef</i>	Pointer to the reference block of specified size.
<i>refStep</i>	Step of the reference block, specifying width of the block in bytes.

<code>pSAD</code>	Pointer to the destination integer.
<code>mcType</code>	MC type <a href="#">IPPVC_MC_APX</a> .

## Description

This function is declared in the `ippvc.h` header file. The function `ippiSAD16x16_8u32s` evaluates the sum of absolute difference between all the elements in current block and the corresponding elements in reference block. The result is stored in integer `pSAD`.

This function is used in the MPEG-2 and H.264 encoders included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsStepErr</code>	Indicates an error when <code>srcCurStep</code> or <code>srcRefStep</code> is less or equal to zero.

---

## SAD16x8

*Evaluates sum of absolute difference between current and reference 16X8 blocks.*

---

## Syntax

```
IppStatus ippiSAD16x8_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const Ipp8u* pSrcRef, int srcRefStep, Ipp32s *pDst, Ipp32s mcType);
```

## Parameters

<code>pSrcCur</code>	Pointer to 16x8 block in the source plane.
<code>srcCurStep</code>	Pitch of the source plane (in bytes).
<code>pSrcRef</code>	Pointer to 16x8 block in the reference plane.
<code>srcRefStep</code>	Pitch of the reference plane (in bytes).
<code>pDst</code>	Pointer to store SAD value.
<code>mcType</code>	MC type <a href="#">IPPVC_MC_APX</a> .

## Description

This function is declared in the `ippvc.h` header file. The function `ippiSAD16x8_8u32s_C1R` evaluates the sum of absolute difference of all the elements in current 16x8 block and the corresponding elements in reference 16x8 block. The result is stored in integer `pDst`.

This function is used in the MPEG-4 encoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## SAD8x8

*Evaluates sum of absolute difference between current and reference 8X8 blocks.*

---

## Syntax

```
IppStatus ippiSAD8x8_8u32s_C1R(const Ipp8u* pSrcCur, int srcCurStep, const
    Ipp8u* pSrcRef, int srcRefStep, Ipp32s* pDst, Ipp32s mcType);
```

## Parameters

<code>pSrcCur</code>	Pointer to 8x8 block in the source plane.
<code>srcCurStep</code>	Pitch of the source plane (in bytes).
<code>pSrcRef</code>	Pointer to 8x8 block in the reference plane.
<code>srcRefStep</code>	Pitch of the reference plane (in bytes).
<code>pDst</code>	Pointer to store SAD value.
<code>mcType</code>	MC type <a href="#">IPPVC MC APX</a> .

## Description

This function is declared in the `ippvc.h` header file. The function `ippiSAD8x8_8u32s_C1R` evaluates the sum of absolute difference of all the elements in current 8x8 block and the corresponding elements in reference 8x8 block. The result is stored in integer `pDst`.

This function is used in the H.261, H.263, H.264 and MPEG-4 encoders included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## SAD4x4

*Evaluates sum of absolute difference between current and reference 4X4 blocks.*

---

### Syntax

```
IppStatus ippISAD4x4_8u32s(const Ipp8u *pSrc, Ipp32s srcStep, const Ipp8u  
    *pRef, Ipp32s refStep, Ipp32s *pSAD, Ipp32s mcType);
```

### Parameters

<code>pSrc</code>	Pointer to 4x4 block in the source plane.
<code>srcStep</code>	Pitch of the source plane (in bytes).
<code>pRef</code>	Pointer to 4x4 block in the reference plane.
<code>refStep</code>	Pitch of the reference plane (in bytes).
<code>pSAD</code>	Pointer to SAD value.
<code>mcType</code>	MC type <a href="#">IPPVC MC APX</a> ; reserved and must be 0.

### Description

This function is declared in the `ippvc.h` header file. The function `ippISAD4x4_8u32s` evaluates the sum of absolute difference of all the elements in current 4x4 block and the corresponding elements in reference 4x4 block. The result is stored in integer `pSAD`.

This function is used in the H.264 encoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## SAD16x16Blocks8x8

*Evaluates four partial sums of absolute differences between current and reference 16X16 blocks.*

---

## Syntax

```
IppStatus ippisAD16x16Blocks8x8_8u16u(const Ipp8u *pSrc, Ipp32s srcStep, const
    Ipp8u *pRef, Ipp32s refStep, Ipp16u *pDstSAD, Ipp32s mcType);
```

## Parameters

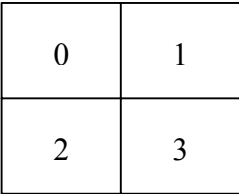
<code>pSrc</code>	Pointer to 16x16 block in the source plane.
<code>srcStep</code>	Pitch of the source plane (in bytes).
<code>pRef</code>	Pointer to 16x16 block in the reference plane.
<code>refStep</code>	Pitch of the reference plane (in bytes).
<code>pDstSAD</code>	Pointer to array of size 4 to store SAD values.
<code>mcType</code>	Reserved and must be 0.

## Description

This function is declared in the `ippvc.h` header file. The function `ippisAD16x16Blocks8x8_8u16u` evaluates the four partial sums of absolute differences of all the elements in current 16x16 block and the corresponding elements in reference 16x16 block. The result is stored in `pRes`.



Figure 16-8     Splitting of 16x16 Block Into Four 8x8 Blocks



This function is used in the H.264 encoder included into IPP Samples. See [introduction](#) to this section.

Return Values

- `ippStsNoErr`Indicates no error.
- `ippStsNullPtrErr`Indicates an error when at least one input pointer is NULL.

SAD16x16Blocks4x4

*Evaluates 16 partial sums of absolute differences between current and reference 16X16 blocks.*

Syntax

```
IppStatus ippisAD16x16Blocks4x4_8u16u(const Ipp8u *pSrc, Ipp32s srcStep, Ipp8u *pRef, Ipp32s refStep, Ipp16u *pDstSAD, Ipp32s mcType);
```

Parameters

- `pSrc`Pointer to 16x16 block in the source plane.
- `srcStep`Pitch of the source plane (in bytes).
- `pRef`Pointer to 16x16 block in the reference plane.
- `refStep`Pitch of the reference plane (in bytes).
- `pDstSAD`Pointer to array of size 16 to store SAD values.

*mcType*

Reserved and must be 0.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiSAD16x16Blocks4x4_8u16u` evaluates 16 partial sums (for each 4x4 block the order shown in [Figure 16-9](#)) of absolute differences between all the elements in current 16x16 block and the corresponding elements in reference 16x16 block. The result is stored in *pRes*.

**Figure 16-9     Splitting of 16x16 Block Into 4x4 Blocks**

---

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

### Return Values

*ippStsNoErr*

Indicates no error.

*ippStsNullPtrErr*

Indicates an error when at least one input pointer is NULL.

---

## FrameFieldSAD16x16

*Calculates SAD between field lines and SAD between frame lines of block 16x16.*

---

### Syntax

```

IppStatus ippiFrameFieldSAD16x16_8u32s_C1R(const Ipp8u *pSrc, int srcStep,
      Ipp32s *pFrameSAD, Ipp32s *pFieldSAD);
IppStatus ippiFrameFieldSAD16x16_16s32s_C1R(const Ipp16s *pSrc, int srcStep,
      Ipp32s *pFrameSAD, Ipp32s *pFieldSAD);

```

## Parameters

<i>pSrc</i>	Pointer to the 16x16 block.
<i>srcStep</i>	Step of the current block, specifying width of the block in bytes.
<i>pFrameSAD</i>	Pointer to the resulting SAD between frame lines of the block.
<i>pFieldSAD</i>	Pointer to the resulting SAD between field lines of the block.

## Description

This function is declared in the `ippvc.h` header file. The functions `ippiFrameFieldSAD16x16_8u32s_C1R` and `ippiFrameFieldSAD16x16_16s32s_C1R` calculate SAD between field lines and between frame lines of 16x16 blocks. The functions are used for decision on DCT type (Frame or Field) in encoding process of interlaced video.

The computation formulas are as follows:

$$pFrameSAD = \sum_{i=0}^6 \sum_{j=0}^{15} |pSrc_{i,j} - pSrc_{i+1,j}| + \sum_{i=8}^{14} \sum_{j=0}^{15} |pSrc_{i,j} - pSrc_{i+1,j}|$$

$$pFieldSAD = \sum_{i=0}^6 \sum_{j=0}^{15} |pSrc_{i*2,j} - pSrc_{i*2+2,j}| +$$

$$+ \sum_{i=0}^6 \sum_{j=0}^{15} |pSrc_{i*2+1,j} - pSrc_{i*2+3,j}|$$

The functions `ippiFrameFieldSAD16x16_8u32s_C1R` and `ippiFrameFieldSAD16x16_16s32s_C1R` are used in the MPEG-4 encoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## Sum of Differences Evaluation

These functions evaluate difference between current and predicted blocks and calculates sums of elements of all residual blocks.

---

### SumsDiff16x16Blocks4x4

*Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 16X16 blocks.*

---

#### Syntax

```
IppStatus ippiSumsDiff16x16Blocks4x4_8u16s_C1(Ipp8u* pSrc, Ipp32s srcStep,
        Ipp8u* pPred, Ipp32s predStep, Ipp16s* pSums, Ipp16s* pDiff);
```

#### Parameters

<i>pSrc</i>	Pointer to a 16x16 block in the current plane.
<i>srcStep</i>	Step of the current plane (in bytes).
<i>pPred</i>	Pointer to a 16x16 block in the reference plane.
<i>predStep</i>	Step of the reference plane (in bytes).
<i>pSums</i>	If not NULL, pointer to an array of size 256 that contains a sequence of 4x4 residual blocks. The array is filled by function.
<i>pDiff</i>	Pointer to an array of size 16 that contains sums of 4x4 difference blocks coefficients. The array is filled by function.

#### Description

This function is declared in the `ippvc.h` header file. The function `ippiSumsDiff16x16Blocks4x4_8u16s_C1` evaluates difference between current and reference 4x4 blocks and calculates sums of 4x4 residual blocks elements in same order as shown in [Figure 16-10](#).

**Figure 16-10 Splitting of 16x16 Block Into Sixteen 4x4 Blocks**

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

$pDiff[(k*4 + l)*16 + i*4 + j] =$

$pSrc[(k*4 + i)*srcStep + (l*4 + j)] - pPred[(k*4 + i)*predStep + (l*4 + j)]$

$k, l, i, j \in [0, 3]$

$$pSums[n] = \sum_{i=0}^{15} pDiff[n*16 + i]$$

$n \in [0, 15]$ .

### Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error when at least one input pointer is NULL.

## SumsDiff8x8Blocks4x4

*Evaluates difference between current and reference 4X4 blocks and calculates sums of 4X4 residual blocks elements for 8X8 blocks.*

---

### Syntax

```
IppStatus ippiSumsDiff8x8Blocks4x4_8u16s_C1(Ipp8u* pSrc, Ipp32s srcStep, Ipp8u* pPred, Ipp32s predStep, Ipp16s* pSums, Ipp16s* pDiff);
```

### Parameters

<i>pSrc</i>	Pointer to an 8x8 block in the current plane.
<i>srcStep</i>	Step of the current plane (in bytes).
<i>pPred</i>	Pointer to an 8x8 block in the reference plane.
<i>predStep</i>	Step of the reference plane (in bytes).
<i>pSums</i>	If not NULL, pointer to an array of size 64 that contains a sequence of 4x4 residual blocks. The array is filled by function.
<i>pDiff</i>	Pointer to an array of size 4 that contains sums of 4x4 difference blocks coefficients. The array is filled by function.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiSumsDiff8x8Blocks4x4_8u16s_C1` evaluates difference between current and reference 4x4 blocks and calculates sums of 4x4 residual blocks elements in same order as shown in [Figure 16-11](#).

**Figure 16-11    Splitting of 8x8 Block Into Sixteen 4x4 Blocks**

---

0	1
2	3

---

$$pDiff[(k*2 + l)*16 + i*4 + j] =$$

$$pSrc[(k*4 + i)*srcStep + (l*4 + j)] - pPred[(k*4 + i)*predStep + (l*4 + j)]$$

$$i, j \in [0, 3]$$

$$k, l \in [0, 1]$$

$$pSums[n] = \sum_{i=0}^{15} pDiff[n*16 + i]$$

$$n \in [0, 3].$$

This function is used in the H.264 encoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## Scanning Functions

Scanning functions convert a two-dimensional 8x8 array into one-dimensional data and vice versa using the predefined scan pattern.

---

### ScanInv

*Performs classical zigzag, alternate-horizontal, or alternate-vertical inverse scan on a block stored in a compact buffer.*

---

#### Syntax

```
ippiScanInv_16s_C1(Ipp16s *pSrc, Ipp16s *pDst, int indxLastNonZero, int scan);
```

## Parameters

<i>pSrc</i>	Pointer to the input block (coefficients in the scan order).
<i>pDst</i>	Pointer to the output block (coefficients in the normal order). The output is in a full buffer that contains 64 elements.
<i>indxLastNonZero</i>	Index of the last non-zero coefficient. Valid within the range of 0 to 63. This parameter provides faster operation. If the value is unknown, set to 63.
<i>scan</i>	Type of the scan, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan.  See the corresponding enumerator on <a href="#">p.16-3</a> .

## Description

The function `ippiScanInv_16s_C1` is declared in the `ippvc.h` header file. This function converts a block of coefficients placed in classical zigzag, alternate-horizontal, or alternate-vertical scan order to a block of coefficients placed in the conventional – left-to-right, top-to-bottom raster scan – order. The zigzag and the two alternate scan patterns are shown, for example, in [\[ITUH263\]](#), Figure14 and [\[ITUH263\]](#), Annex I, Figure I.2.

This function is used in the H.263 decoder included into IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm> .

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <i>indxLastNonZero</i> is out of the range [0, 63].



---

## ScanFwd

*Performs classical zigzag, alternate-horizontal, or alternate-vertical forward scan on a block.*

---

### Syntax

```
ippiScanFwd_16s_C1(Ipp16s *pSrc, Ipp16s *pDst, int countNonZero, int scan);
```

### Parameters

<i>pSrc</i>	Pointer to the input block (coefficients in the normal order).
<i>pDst</i>	Pointer to the output block (coefficients in the scan order).
<i>countNonZero</i>	Number of non-zero coefficients in the block. Valid within the range of 1 to 64. This parameter provides faster operation. If the value is unknown, set to 64.
<i>scan</i>	Type of the scan, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan.  See the corresponding enumerator on <a href="#">p.16-3</a> .

### Description

The function `ippiScanFwd_16s_C1` is declared in the `ippvc.h` header file. This function converts a block of coefficients placed in the conventional – left-to-right, top-to-bottom raster scan – order to a block of coefficients placed in classical zigzag, alternate-horizontal, or alternate-vertical scan order. See [Figure 16-17](#) for the scanning process in line with the classical zigzag pattern. The zigzag and the two alternate scan patterns are shown, for example, in [\[ITUH263\]](#), Figure 14 and [\[ITUH263\]](#), Annex I, Figure I.2.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <i>countNonZero</i> is out of the range [1, 64].

---

## **ZigzagInvClassical\_Compact, ZigzagInvHorizontal\_Compact, ZigzagInvVertical\_Compact**

*Perform classical, horizontal, or vertical inverse zigzag scan on a block stored in a compact buffer.*

---

### **Syntax**

```
IppStatus ippiZigzagInvClassical_Compact_16s(const Ipp16s* pSrc,  
                                             int len, Ipp16s* pDst);  
IppStatus ippiZigzagInvHorizontal_Compact_16s(const Ipp16s* pSrc,  
                                              int len, Ipp16s* pDst);  
IppStatus ippiZigzagInvVertical_Compact_16s(const Ipp16s* pSrc,  
                                             int len, Ipp16s* pDst);
```

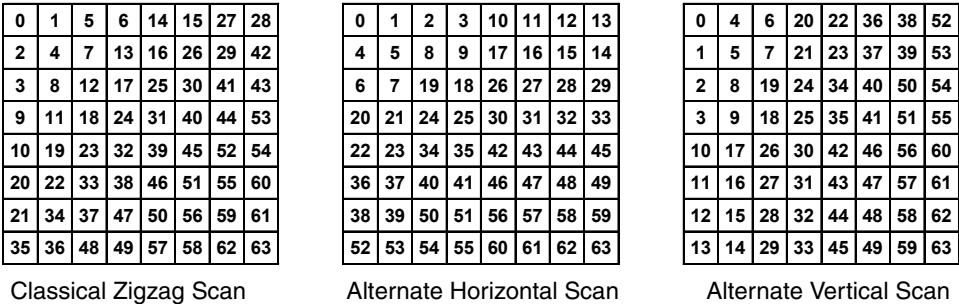
### **Parameters**

<i>pSrc</i>	Pointer to input (zigzagged) block.
<i>pDst</i>	Pointer to output (normally scanned) block. The output is in a full buffer which contains 64 elements.
<i>len</i>	Length of the input and output compact buffer.

### **Description**

The functions `ippiZigzagInvClassical_Compact_16s`, `ippiZigzagInvHorizontal_Compact_16s`, and `ippiZigzagInvVertical_Compact_16s` are declared in the `ippalign.h` file. These functions reorder classical, horizontal, or vertical inverse zigzag scan respectively to the normal order on a block stored in a compact buffer. [Figure 16-12](#) shows three zigzagged scan patterns. Numbers within cells indicate the scanning sequence.

Figure 16-12    Scan Patterns



An inverse scan is a mapping from the normal scan pattern to one zigzag scan pattern. For example, after performing a classical zigzag scan on a block,  $pDst[3] = pSrc[6]$ . See the description of [ScanInv](#) function for more details.



**NOTE.** The output buffer  $pDst$  should be zeroed out prior to the function call.

Return Values

- ippStsNoErr
- Indicates no error.
- ippStsNullPtrErr
- Indicates an error condition if at least one of the specified pointers is NULL.

## Color Conversion

---

### CbYCr422ToYCbCr420\_Rotate

*Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image with rotation.*

---

#### Syntax

```
IppStatus ippCbYCr422ToYCbCr420_Rotate_8u_C2P3R(const Ipp8u* pSrc, int
    srcStep, IppiSize srcRoi, Ipp8u *pDst[3], int dstStep[3], int rotation);
IppStatus ippCbYCr422ToYCbCr420_Rotate_8u_P3R(const Ipp8u* pSrc[3], int
    srcStep[3], IppiSize srcRoi, Ipp8u *pDst[3], int dstStep[3], int rotation);
```

#### Parameters

<i>pSrc</i>	Pointer to the source image for pixel-order data. Array of pointers to the separate source image planes for planar data.
<i>srcStep</i>	Step in bytes through the source image. Array of step values in bytes through the source image.
<i>srcRoi</i>	ROI of the source image, in pixels. The ROI of the destination image is calculated by user.
<i>pDst</i>	Array of pointers to the destination image planes.
<i>dstStep</i>	Array of step values through the destination image planes .
<i>rotation</i>	Rotation control parameter. Possible values: <div style="margin-left: 40px;"> IPPVC_ROTATE_DISABLE no rotation  IPPVC_ROTATE_90CCW rotating by 90° counterclockwise  IPPVC_ROTATE_90CW rotating by 90° clockwise  IPPVC_ROTATE_180 rotating by 180° </div> See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.

## Description

The function `ippiCbYCr422ToYCbCr420_Rotate` is declared in the `ippvc.h` file. This function converts [4:2:2](#) two-channel or three-plane CbYCr image *pSrc* to the [4:2:0](#) YCbCr three-plane image *pDst*. The two-channel source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, .... Three-plane source image has the following order of pointers: Cb-plane, Y-plane, Cr-plane. The destination image has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see [Table 6-2](#) and [Table 6-3](#)).

The function additionally rotates or flips an image in accordance with the value of the parameter *rotation*.

[Example 16-2](#) shows how to call `ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R`.

### Example 16-2 Call of `ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R`

```
#define ROTATE IPPVC_ROTATE_90CCW
{
    Ipp8u* pSrcIm;
    Ipp8u* pDstIm[3];
    int    stepSrc;
    int    stepDst[3];
    IppiSize srcRoi = {32,32};
    IppiSize dstRoi;

    switch( ROTATE )
    {
        case IPPVC_ROTATE_DISABLE:
        case IPPVC_ROTATE_180:
            dstRoi.width  = srcRoi.width ;
            dstRoi.height = srcRoi.height;
            break;
        case IPPVC_ROTATE_90CCW:
        case IPPVC_ROTATE_90CW:
            dstRoi.width  = srcRoi.height;
            dstRoi.height = srcRoi.width ;
            break;
    }
    pSrcIm    = ippiMalloc_8u_C2(srcRoi.width, srcRoi.height, &stepSrc);
    pDstIm[0] = ippiMalloc_8u_C1(dstRoi.width, dstRoi.height,
    &(stepDst[0]));
    pDstIm[1] = ippiMalloc_8u_C1(dstRoi.width/2, dstRoi.height/2,
    &(stepDst[1]));
    pDstIm[2] = ippiMalloc_8u_C1(dstRoi.width/2, dstRoi.height/2,
    &(stepDst[2]));
    ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R(pSrcIm, stepSrc, srcRoi,
    pDstIm, stepDst, ROTATE);
    ippiFree(pDstIm[0]);
    ippiFree(pDstIm[1]);
    ippiFree(pDstIm[2]);
}
```

## Example 16-2 Call of `ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R` (continued)

---

```
    ippiFree(pSrcIm);
}
```

---

### Return Values

<code>ippiStsNoErr</code>	Indicates no error. Any other value indicates an error.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if any of the specified pointers is NULL.
<code>ippiStsSizeErr</code>	Indicates an error condition if any field of the <code>srcRoi</code> is less than 2.
<code>ippiStsDoubleSize</code>	Indicates the condition when the values <code>srcRoi.width</code> and <code>srcRoi.height</code> are not multiples of 2. The function reduces their original values to the nearest multiples of 2 and continues operation.




---

**NOTE.** If `srcRoi.width` and `srcRoi.height` are not multiples of 2, the function reduces the values to the nearest multiples of 2. For example, if the size value is 7, the function approximates this value to 6.

---



---

## ResizeCCRotate

*Creates a low-resolution preview image for high-resolution video or still capture applications.*

---

### Syntax

```
IppStatus ippiResizeCCRotate_8u_C2R(const Ipp8u *pSrc, int srcStep, IppiSize
    srcRoi, Ipp16u *pDst, int dstStep, int zoomFactor, int interpolation, int
    colorConversion, int rotation);
```

### Parameters

<code>pSrc</code>	Pointer to the source image. Input byte order is Cb Y Cr Y.
<code>srcStep</code>	Step in bytes through the source image.
<code>srcRoi</code>	ROI of the source image, in pixels. The ROI of the destination image is calculated by user.

<i>pDst</i>	Pointer to the destination image. As an output parameter, indicates a pointer to the start of the buffer containing the resized and rotated image with completed color conversion.
<i>dstStep</i>	Step in bytes through the destination image.
<i>zoomFactor</i>	Parameter, indicating downscale factor; takes values 2, 4 or 8 for 2:1, 4:1, and 8:1 downscale respectively.
<i>interpolation</i>	Type of interpolation to perform resampling of the input image. The following types are currently supported: IPPI_INTER_NN               nearest neighbor interpolation IPPI_INTER_LINEAR          linear interpolation.
<i>colorConversion</i>	Color conversion control parameter, must be set to one of the following pre-defined values: IPPVC_CbYCr422ToBGR565 IPPVC_CbYCr422ToBGR555.  See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.
<i>rotation</i>	Rotation control parameter. Possible values: IPPVC_ROTATE_DISABLE   no rotation IPPVC_ROTATE_90CCW    rotating by 90° counterclockwise IPPVC_ROTATE_90CW     rotating by 90° clockwise IPPVC_ROTATE_180      rotating by 180°  See the corresponding <a href="#">enumerator</a> in the introduction to the General Functions section.

## Description

The function `ippiResizeCCRotate_8u_C2R` is declared in the `ippvc.h` file. This function synthesizes a low-resolution preview image for high-resolution video or still capture applications. The function combines scale reduction 2:1, 4:1 or 8:1, color space conversion, and rotation of an image.

[Example 16-3](#) shows how to call `ippiResizeCCRotate_8u_C2R`.

### Example 16-3 Call of `ippiResizeCCRotate_8u_C2R`

```
#define ROTATE IPPVC_ROTATE_90CW
#define ZOOMOUT 8
```

**Example 16-3** Call of `ippiResizeCCRotate_8u_C2R` (continued)

---

```
{
    Ipp8u*  pSrcIm;
    Ipp16u* pDstIm;
    int     stepSrc;
    int     stepDst;
    IppiSize srcRoi = {64,64};
    IppiSize dstRoi;
    /* you should calculate dstROI for memory allocation only. */
    /* dstROI is a function of parameters zoomFactor and rotation. */
    switch (ROTATE){
        case IPPVC_ROTATE_DISABLE:
        case IPPVC_ROTATE_180:
            dstRoi.width  = srcRoi.width /ZOOMOUT;
            dstRoi.height = srcRoi.height/ZOOMOUT;
            break;
        case IPPVC_ROTATE_90CCW:
        case IPPVC_ROTATE_90CW:
            dstRoi.width  = srcRoi.height/ZOOMOUT;
            dstRoi.height = srcRoi.width /ZOOMOUT;
            break;
    }
    pSrcIm = ippiMalloc_8u_C2 (srcRoi.width, srcRoi.height, &stepSrc);
    pDstIm = ippiMalloc_16u_C1(dstRoi.width, dstRoi.height, &stepDst);
    ippiResizeCCRotate_8u_C2R (pSrcIm, stepSrc, srcRoi, pDstIm, stepDst,
    ZOOMOUT, IPP_I_INTER_NN, IPPVC_CbYCr422ToBGR555, ROTATE);
    ippiFree(pDstIm);
    ippiFree(pSrcIm);
}
```

---

**Return Values**

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is <code>NULL</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>srcRoi.width</code> or <code>srcRoi.height</code> is less than <code>scaleFactor</code> .
<code>ippStsInterpolationErr</code>	Indicates an invalid value of <i>interpolation</i> control parameter.
<code>ippStsResizeFactorErr</code>	Indicates an invalid value of <i>zoomFactor</i> control parameter.
<code>ippStsBadArgErr</code>	Indicates an invalid value of <i>rotation</i> control parameter.



ippStsDoubleSize

Indicates the condition when the values `srcRoi.width` and `srcRoi.height` are not multiples of 2. The function reduces their original values to the nearest multiples of 2 and continues operation.



**NOTE.** If `srcRoi.width` and `srcRoi.height` are not multiples of 2, the function reduces the values to the nearest multiples of 2. For example, if the size value is 7, the function approximates this value to 6.

## Video Processing

This section describes a function that is used to process decompressed video data.

---

## DeinterlaceFilterTriangle

*Deinterlaces video plane.*

---

### Syntax

```
IppStatus ippDeinterlaceFilterTriangle_8u_C1R(const Ipp8u *pSrc, Ipp32s
srcStep, Ipp8u *pDst, Ipp32s dstStep, IppiSize roiSize, Ipp32u centerWeight,
Ipp32u layout);
```

### Parameters

<code>pSrc</code>	Pointer to the source video plane.
<code>srcStep</code>	Step through the source video plane.
<code>pDst</code>	Pointer to the destination video plane.
<code>dstStep</code>	Step through the destination video plane.
<code>roiSize</code>	Size of ROI; height should be greater than 3.
<code>centerWeight</code>	Weight of filtered pixel, must lie within the range from 0 to 256.

*layout* Plane layout, required when the plane is only a part of the frame. Takes the following values:

- IPP\_UPPER for the first slice
- IPP\_CENTER for the middle slices
- IPP\_LOWER for the last slice
- IPP\_LOWER&&IPP\_UPPER&&IPP\_CENTER for the image that is not sliced.

## Description

The function `ippiDeinterlaceFilterTriangle_8u_C1R` is declared in the `ippvc.h` header file. This function deinterlaces video plane. The function performs triangle filtering of the image to remove interlacing flicker effect that arises when analogue interlaced TV data is viewed on a computer monitor.

Pixels are filtered by the following formula:

$$\text{pixel\_new} = [\text{pixel\_prev} * (256 - \text{centerWeight}) / 2 + \text{pixel} * \text{centerWeight} + \text{pixel\_next} * (256 - \text{centerWeight}) / 2] / 256.$$

When *layout* takes the value of IPP\_CENTER or IPP\_LOWER, the last data row from the previous slice should be accessible, that is, `pSrc[-srcStep]` should also be a valid pointer.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsSizeErr</code>	Indicates an error when <i>roiSize</i> has a field with zero or negative value.
<code>ippStsBadArgErr</code>	Indicates invalid argument.

## MPEG-1 and MPEG-2

This section contains `ippVC` functions for encoding and decoding of video data according to MPEG-1([ISO11172](#)) and MPEG-2([ISO13818](#)) standards.

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm> .

### Structures and Enumerations

The enumeration `IPPVC_MV_TYPE` indicates motion vector types (`FIELD`, `FRAME`).

```
typedef enum _IPPVC_MV_TYPE{
    IPPVC_MV_FIELD =      0x0,
    IPPVC_MV_FRAME =      0x1,
} IPPVC_ MV_TYPE;
```

**Table 16-11     MPEG-1 and MPEG-2 Video Decoding Functions**

Function Short Name	Description
<b>Variable Length Decoding Functions</b>	
<a href="#">HuffmanTableInitAlloc</a>	Allocates memory and initializes a table that contains codes for macroblock address increment, macroblock type, macroblock pattern, or motion vectors.
<a href="#">HuffmanRunLevelTableInitAlloc</a>	Allocates memory and initializes a table that contains codes for DCT coefficients, that is, Run-Level codes.
<a href="#">DecodeHuffmanOne</a>	Decodes a code from the bitstream using a specified table.
<a href="#">DecodeHuffmanPair</a>	Decodes one code using specified table and gets two decoded values.
<a href="#">ReconstructDCTBlock_MPEG1</a>	Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.
<a href="#">ReconstructDCTBlockIntra MPEG1</a>	Decodes 8X8 intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.

**Table 16-11 MPEG-1 and MPEG-2 Video Decoding Functions (continued)**

Function Short Name	Description
<a href="#"><u>ReconstructDCTBlock MPEG2</u></a>	Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.
<a href="#"><u>ReconstructDCTBlockIntra MPEG2</u></a>	Decodes 8X8 intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.
<a href="#"><u>HuffmanTableFree</u></a>	Frees memory allocated for VLC table.
<b>Inverse Quantization Functions</b>	
<a href="#"><u>QuantInvIntra MPEG2</u></a>	Performs inverse quantization for intra frames according to MPEG-2 standard.
<a href="#"><u>QuantInv MPEG2</u></a>	Performs inverse quantization for non-intra frames according to MPEG-2 standard.
<b>Inverse Discrete Cosine Transformation</b>	
<a href="#"><u>DCT8x8Inv AANTransposed 16s C1R</u></a>	Performs inverse DCT on pre-transposed block.
<a href="#"><u>DCT8x8Inv AANTransposed 16s8u C1R</u></a>	Performs inverse DCT on pre-transposed block and converts output to unsigned char format.
<a href="#"><u>DCT8x8Inv AANTransposed 16s P2C2R</u></a>	Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one array.
<a href="#"><u>DCT8x8Inv AANTransposed 16s8u P2C2R</u></a>	Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one unsigned char array.
<b>Motion Compensation Functions</b>	
<a href="#"><u>MC16x16</u></a>	Performs motion compensation for a predicted 16X16 block.
<a href="#"><u>MC16x8</u></a>	Performs motion compensation for a predicted 16X8 block.
<a href="#"><u>MC8x16</u></a>	Performs motion compensation for a predicted 8X16 block.
<a href="#"><u>MC8x8</u></a>	Performs motion compensation for a predicted 8X8 block.
<a href="#"><u>MC8x4</u></a>	Performs motion compensation for a predicted 8X4 block.
<a href="#"><u>MC4x8</u></a>	Performs motion compensation for a predicted 4X8 block.

**Table 16-11**    **MPEG-1 and MPEG-2 Video Decoding Functions (continued)**

Function Short Name	Description
<a href="#">MC16x4</a>	Performs motion compensation for predicted UV 16X4 block.
<a href="#">MC16x8UV</a>	Performs motion compensation for predicted UV 16X8 block.
<a href="#">MC16x16B</a>	Performs motion compensation for a bi-predicted 16X16 block.
<a href="#">MC16x8B</a>	Performs motion compensation for a bi-predicted 16X8 block.
<a href="#">MC8x16B</a>	Performs motion compensation for a bi-predicted 8X16 block.
<a href="#">MC8x8B</a>	Performs motion compensation for a bi-predicted 8X8 block.
<a href="#">MC8x4B</a>	Performs motion compensation for a bi-predicted 8X4 block.
<a href="#">MC16x4B</a>	Performs motion compensation for bi-predicted UV 16X4 block.
<a href="#">MC16x8UV</a>	Performs motion compensation for bi-predicted UV 16X8 block.

**Table 16-12**    **MPEG-1 and MPEG-2 Video Encoding Functions**

Function Short Name	Description
<b>Motion Estimation Functions</b>	
<a href="#">GetDiff16x16</a>	Evaluates difference between current predicted and reference blocks of 16x16 elements.
<a href="#">GetDiff16x8</a>	Evaluates difference between current predicted and reference blocks of 16x8 elements.
<a href="#">GetDiff8x8</a>	Evaluates difference between current predicted and reference blocks of 8x8 elements.
<a href="#">GetDiff8x16</a>	Evaluates difference between current predicted and reference blocks of 8x16 elements.
<a href="#">GetDiff8x4</a>	Evaluates difference between current predicted and reference blocks of 8x4 elements.

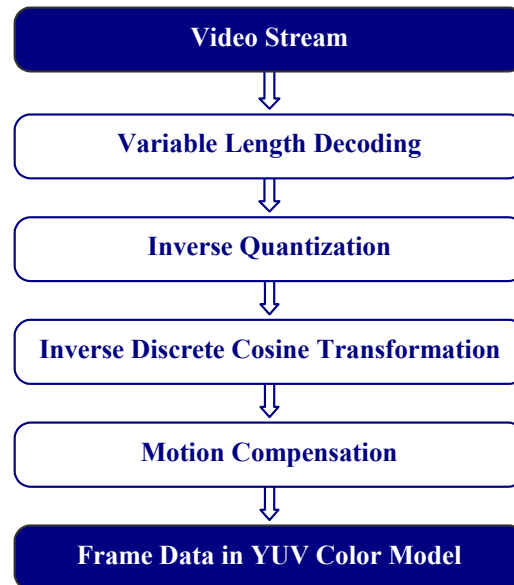
**Table 16-12 MPEG-1 and MPEG-2 Video Encoding Functions (continued)**

Function Short Name	Description
<a href="#"><u>GetDiff16x16B</u></a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 16x16 elements.
<a href="#"><u>GetDiff16x8B</u></a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 16x8 elements.
<a href="#"><u>GetDiff8x8B</u></a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x8 elements.
<a href="#"><u>GetDiff8x16B</u></a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x16 elements.
<a href="#"><u>GetDiff8x4B</u></a>	Evaluates difference between current bi-predicted and mean of two reference blocks of 8x4 elements.
<a href="#"><u>SqrDiff16x16</u></a>	Evaluates sum of squares of differences between current and reference blocks.
<a href="#"><u>SqrDiff16x16B</u></a>	Evaluates the sum of squares of differences between the current bi-predicted block and the mean of two reference blocks.
<a href="#"><u>VarMean8x8</u></a>	Evaluates variance and mean of 8X8 block.
<a href="#"><u>VarMeanDiff16x16</u></a>	Evaluates variances and means of four 8x8 blocks of difference between two 16x16 blocks.
<a href="#"><u>VarMeanDiff16x8</u></a>	Evaluates variances and means of four 8x8 blocks of difference between two 16x8 blocks.
<a href="#"><u>SAD16x16</u></a>	Evaluates sum of absolute difference between current and reference blocks.
<b>Quantization Functions</b>	
<a href="#"><u>QuantIntra MPEG2</u></a>	Performs quantization on DCT coefficients for intra blocks in-place with specified quantization matrix according to MPEG-2 standard.
<a href="#"><u>Quant MPEG2</u></a>	Performs quantization on DCT coefficients for non-intra blocks in-place with specified quantization matrix according to MPEG-2 standard.
<b>Huffman Encoding Functions</b>	
<a href="#"><u>CreateRLEncodeTable</u></a>	Creates Run-Level Encode Table.
<a href="#"><u>PutIntraBlock</u></a>	Encodes, rearranges and puts intra block into the bit stream.
<a href="#"><u>PutNonIntraBlock</u></a>	Encodes, rearranges and puts non-intra block into the bit stream.

## Video Data Decoding

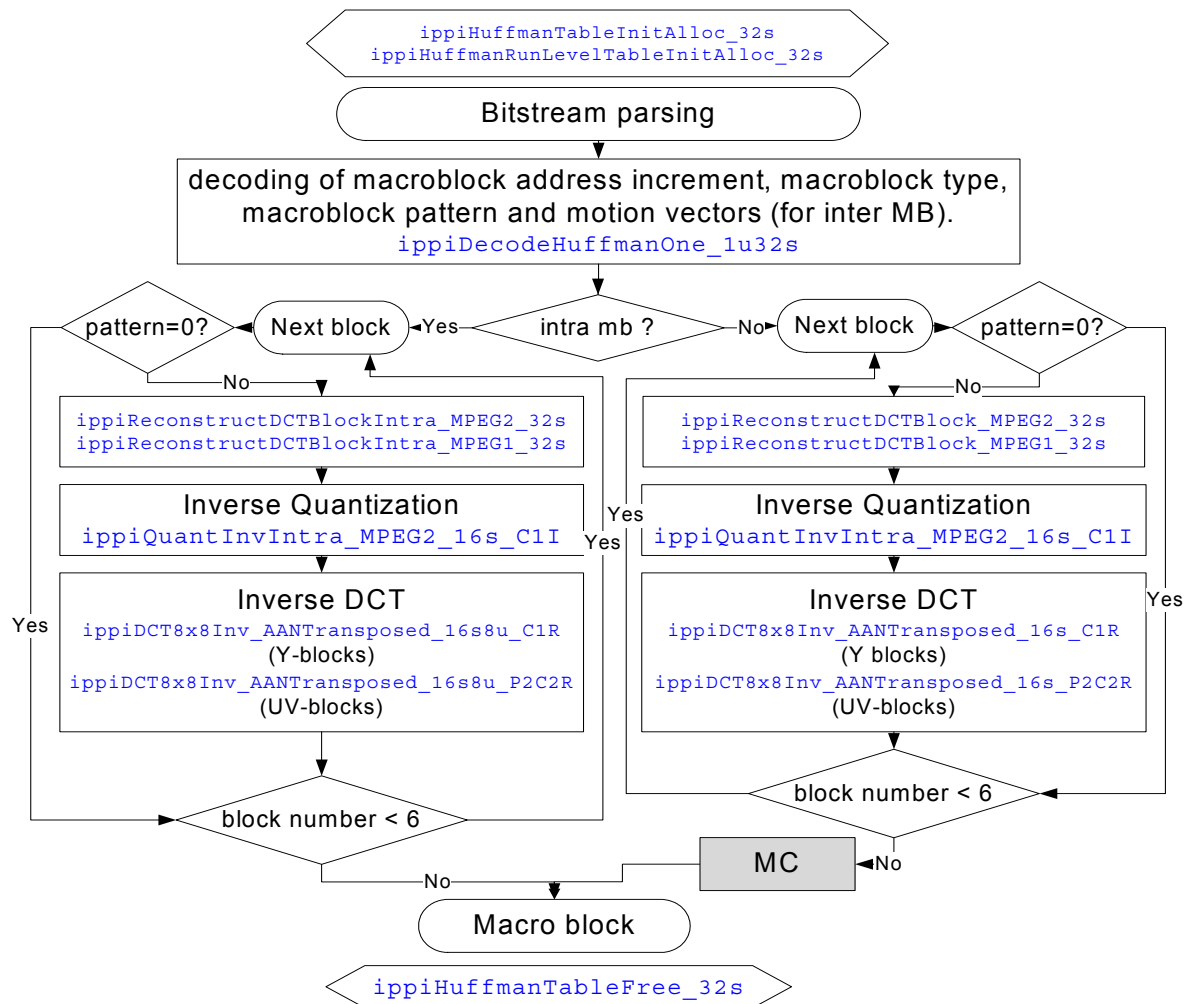
This section describes principal steps of MPEG-1 and MPEG-2 video decoding in accordance with the standard decoding pipeline shown in [Figure 16-13](#).

**Figure 16-13** Standard Decoding Pipeline



**NOTE.** The inverse DCT is one of the most common transformations. It may be performed using Ipp image processing functions: `ippiDCT8x8Inv_16s_C1R` for 8x8 blocks of non-intra type and `ippiDCT8x8Inv_16s8u_C1R` for 8x8 blocks of intra type.

**Figure 16-14 Macroblock Decoding Scheme**

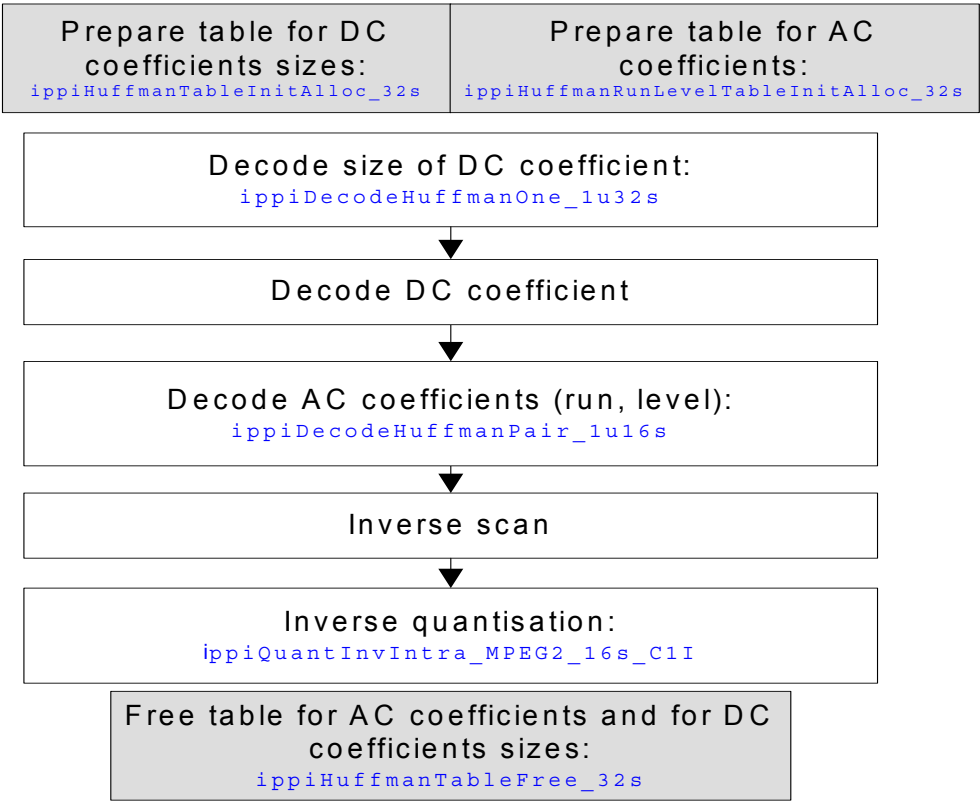






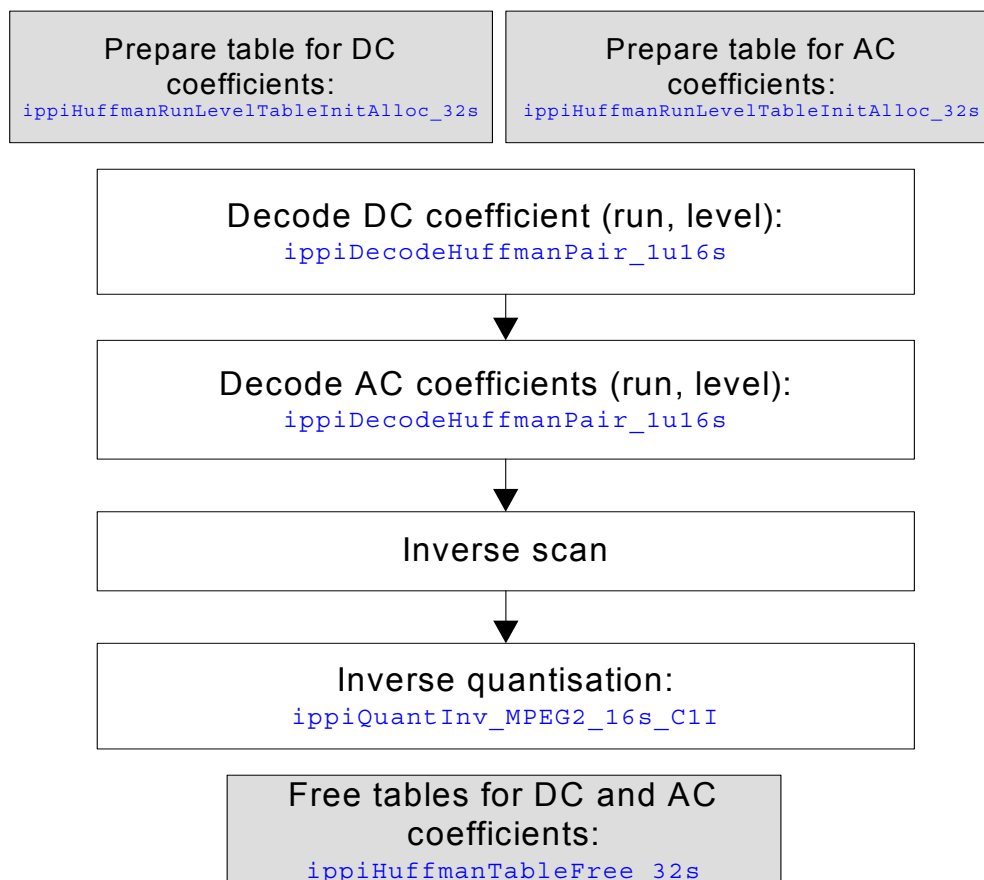
**NOTE.** High-level functions `ippiReconstructDCTBlockIntra_MPEG2_32s` and `ippiReconstructDCTBlockIntra_MPEG1_32s` may be replaced by low-level functions according to the scheme presented in [Figure 16-15](#) and high-level functions `ippiReconstructDCTBlock_MPEG2_32s` and `ippiReconstructDCTBlock_MPEG1_32s` may be replaced by low-level functions according to the scheme presented in [Figure 16-16](#).

Figure 16-15 DCT Intra Block Reconstruction



**Figure 16-16 DCT Non-Intra Block Reconstruction**

---



### Variable Length Decoding

Video data in the bitstream is encoded with Variable Length Code (VLC) tables so that the shortest codes correspond to the most frequent values and the longer codes correspond to the less frequent values. MPEG standard tables contain possible codes and their values.

## Decoding with Run-Level Tables

These functions decode, rearrange, and inverse quantize non-intra and intra 8x8 blocks of DCT coefficients using Run-Level Tables.

---

## ReconstructDCTBlock\_MPEG1

*Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.*

---

### Syntax

```
IppStatus ippiReconstructDCTBlock_MPEG1_32s(Ipp32u **ppBitStream, int
    *pOffset, const Ipp32s *pDCSizeTable, const Ipp32s *pACTable, Ipp32s*
    pScanMatrix, int QP, Ipp16s *pQPMatrix, Ipp16s *pDstBlock, Ipp32s
    *pDstSize);
```

### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bit stream.
<i>pOffset</i>	Pointer to the offset between the bit that <i>ppBitStream</i> points to and the start of the code.
<i>pDCSizeTable</i>	Pointer to the table containing codes for DC coefficient, that is, the first of the DCT coefficients.
<i>pACTable</i>	Pointer to the table containing Run-Level codes for all DCT coefficients except the first one.
<i>pScanMatrix</i>	Pointer to the matrix containing indices of elements in scanning sequence.
<i>QP</i>	Quantizer scale factor read from the bit stream.
<i>pQPMatrix</i>	Pointer to the weighting matrix imposed by standard or user-defined.
<i>pDstBlock</i>	Pointer to decoded elements.
<i>pDstSize</i>	Position of the last non-zero block coefficient in scanning sequence.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiReconstructDCTBlock_MPEG1_32s` is applied to predicted and bi-predicted blocks and processes all the DCT coefficients in block using the tables of Run-Level codes: `pDCSizeTable` for DC coefficient and `pACTable` for AC coefficients.

The function uses the tables derived with [HuffmanRunLevelTableInitAlloc](#) function.

The pointer `pScanMatrix` points to the scanning matrix described in MPEG standard. [Figure 16-17](#) illustrates a simple matrix and a sequence. After decoding, data is rearranged from scanning sequence to linear sequence. Then inverse quantization is performed: each of the DCT coefficients is multiplied by a corresponding value from the weighting matrix `pQPMatrix` and by the quantizer scale factor, which are read from the bit stream.

The function decodes the block of 8x8 DCT coefficients. The result is sent to `pDstBlock` and `*pDstSize` is the position of the last non-zero coefficient. After decoding, the pointers to code in the bitstream are reset as shown in [Figure 16-5](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsH263VLCCodeErr</code>	Indicates an error when decoding in accordance with H263 standard.

---

## ReconstructDCTBlockIntra\_MPEG1

*Decodes 8X8 intra block using table of Run-Level codes for standard MPEG1, rearranges and performs inverse quantization.*

---

## Syntax

```
IppStatus ippiReconstructDCTBlockIntra_MPEG1_32s(Ipp32u **ppBitStream, int
    *pOffset, const Ipp32s *pDCSizeTable, const Ipp32s *pACTable, Ipp32s*
    pScanMatrix, int QP, Ipp16s *pQPMatrix, Ipp16s *pDCPred, Ipp16s *pDstBlock,
    Ipp32s *pDstSize);
```

## Parameters

<code>ppBitStream</code>	Double pointer to the current position in the bit stream.
--------------------------	---

<i>pOffset</i>	Pointer to the offset between the bit that <i>ppBitStream</i> points to and the start of the code.
<i>pDCSizeTable</i>	Pointer to the table containing codes for DC coefficient, that is, the first of the DCT coefficients.
<i>pACTable</i>	Pointer to the table containing Run-Level codes for all DCT coefficients except the first one.
<i>pScanMatrix</i>	Pointer to the scanning matrix imposed by standard or user-defined.
<i>QP</i>	Quantizer scale factor read from the bit stream.
<i>pQPMatrix</i>	Pointer to the weighting matrix imposed by standard or user-defined.
<i>pDCPred</i>	Pointer to the value to be added to the DC coefficient. This value is read from the special table imposed by standard.
<i>pDstBlock</i>	Pointer to the decoded elements.
<i>pDstSize</i>	Position of the last non-zero block coefficient in scanning sequence.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiReconstructDCTBlockIntra_MPEG1_32s` is applied to intra blocks and processes all the DCT coefficients in block using *pACTable*, which contains Run-Level codes for AC coefficients, except for DC coefficient that is to be processed using *pDCTable*.

The function uses the tables derived with [HuffmanTableInitAlloc](#) and [HuffmanRunLevelTableInitAlloc](#) functions.

The pointer *pScanMatrix* points to the scanning matrix described in MPEG standard. [Figure 16-17](#) illustrates a simple matrix and a sequence. After decoding, data is rearranged from scanning sequence to linear sequence. Then inverse quantization is performed: each of the DCT coefficients is multiplied by a corresponding value from the weighting matrix *pQPMatrix* and by the quantizer scale factor, which are read from the bit stream.

The DC coefficient is increased by *pDCPred*. After performing the function, the *pDCPred* stores the sum of its initial value and DC coefficient.

The function decodes the block of 8x8 DCT coefficients. The result is sent to *pDstBlock* and *\*pDstSize* is the position of the last non-zero coefficient. After decoding, the pointers to code in the bitstream are reset as shown in [Figure 16-5](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsH263VLCCodeErr</code>	Indicates an error when decoding in accordance with H263 standard.

---

## ReconstructDCTBlock\_MPEG2

*Decodes 8X8 non-intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.*

---

### Syntax

```
IppStatus ippIReconstructDCTBlock_MPEG2_32s(Ipp32u **ppBitStream, int
*pOffset, const IppVCHuffmanSpec_32s *pDCTable, const IppVCHuffmanSpec_32s
*pACTable, Ipp32s *pScanMatrix, int QP, Ipp16s *pQPMatrix, Ipp16s
*pDstBlock, Ipp32s *pDstSize);
```

### Parameters

<code>ppBitStream</code>	Double pointer to the current position in the bit stream.
<code>pOffset</code>	Pointer to the offset between the bit that <code>ppBitStream</code> points to and the start of the code.
<code>pDCTable</code>	Pointer to the table containing codes for DC coefficient, that is, the first of the DCT coefficients.
<code>pACTable</code>	Pointer to the table containing Run-Level codes for all DCT coefficients except the first one.
<code>pScanMatrix</code>	Pointer to the matrix containing indices of elements in scanning sequence.
<code>QP</code>	Quantizer scale factor read from the bit stream.
<code>pQPMatrix</code>	Pointer to the weighting matrix imposed by standard or user-defined; may be NULL.
<code>pDstBlock</code>	Pointer to decoded elements.
<code>pDstSize</code>	Position of the last non-zero block coefficient in scanning sequence.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiReconstructDCTBlock_MPEG2_32s` decodes an 8x8 block using tables of Run-Level codes, rearranges the block and performs inverse quantization. The function is applied to predicted and bi-predicted blocks and processes all the DCT coefficients in block.

The function uses the tables derived with [HuffmanRunLevelTableInitAlloc](#) function.

The function decodes the block of 8x8 DCT coefficients. Simultaneous processing of the coefficients enhances performance. The result is sent to `pDstBlock` and `*pDstSize` is the position of the last non-zero coefficient. After decoding, the pointers to code in the bitstream are reset as shown in [Figure 16-5](#).

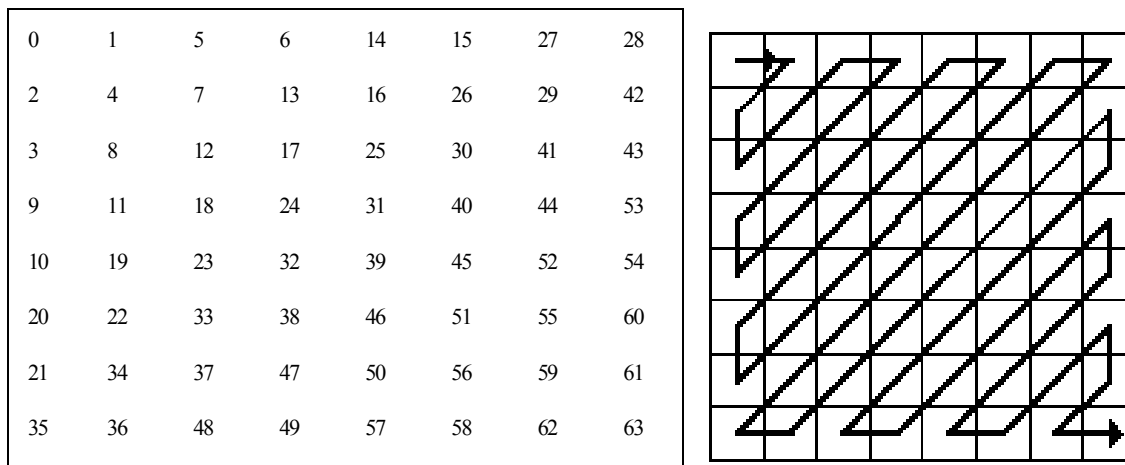
The pointer `pScanMatrix` points to one of the pre-defined scanning matrices, depending on the bitstream. [Figure 16-17](#) illustrates a simple matrix and a sequence. After decoding, data is rearranged from scanning sequence to linear sequence.

Then inverse quantization is performed: each of the DCT coefficients is multiplied by a corresponding value from the weighting matrix `pQPMatrix` and by the quantizer scale factor, which are read from the bit stream. If `pQPMatrix` is NULL, the default matrix `non_intra_quantizer_matrix` is used, which is described in subclause 6.3.11 of [\[ISO13818-2\]](#).

This function is used in the MPEG-2 decoder included into IPP Samples. See [introduction](#) to this section.

**Figure 16-17 Scanning Matrix and Sequence**

---



### Return Values

- `ippStsNoErr` Indicates no error.
- `ippStsNullPtrErr` Indicates an error when at least one input pointer is NULL.
- `ippStsH263VLCCodeErr` Indicates an error when decoding in accordance with H263 standard.



---

## ReconstructDCTBlockIntra\_MPEG2

*Decodes 8X8 intra block using table of Run-Level codes for standard MPEG2, rearranges and performs inverse quantization.*

---

### Syntax

```
IppStatus ippiReconstructDCTBlockIntra_MPEG2_32s(Ipp32u **ppBitStream, int
    *pOffset, const IppVCHuffmanSpec_32s *pDCSizeTable, const
    IppVCHuffmanSpec_32s *pACTable, Ipp32s *pScanMatrix, int QP, Ipp16s
    *pQPMatrix, Ipp16s *pDCPred, Ipp32s shiftDCVal, Ipp16s *pDstBlock, Ipp32s
    *pDstSize);
```

### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bit stream.
<i>pOffset</i>	Pointer to the offset between the bit that <i>ppBitStream</i> points to and the start of the code.
<i>pDCSizeTable</i>	Pointer to the table containing codes for DC coefficient, that is, the first of the DCT coefficients.
<i>pACTable</i>	Pointer to the table containing Run-Level codes for all DCT coefficients except the first one.
<i>pScanMatrix</i>	Pointer to the scanning matrix imposed by standard or user-defined.
<i>QP</i>	Quantizer scale factor read from the bit stream.
<i>pQPMatrix</i>	Pointer to the weighting matrix imposed by standard or user-defined.
<i>pDCPred</i>	Pointer to the value to be added to the DC coefficient. This value is read from the special table imposed by standard.
<i>shiftDCVal</i>	Integer value. The DC coefficient must be multiplied by $2^{\text{shiftDCVal}}$ .
<i>pDstBlock</i>	Pointer to the decoded elements.
<i>pDstSize</i>	Position of the last non-zero block coefficient in scanning sequence.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiReconstructDCTBlockIntra_MPEG2_32s` decodes an 8x8 intra block using tables of Run-Level codes, rearranges the block and performs inverse quantization. The function is applied to intra blocks and processes all the DCT coefficients in block, except for DC coefficient that is to be processed using the table `pDCSizeTable`.

The function uses the tables derived with [HuffmanTableInitAlloc](#) and [HuffmanRunLevelTableInitAlloc](#) functions.

The function decodes the block of 8x8 DCT coefficients. Simultaneous processing of the coefficients enhances performance. The result is sent to `pDstBlock` and `*pDstSize` is the position of the last non-zero coefficient. After decoding, the pointers to code in the bitstream are reset as shown in [Figure 16-5](#).

The pointer `pScanMatrix` points to the scanning matrix that is read from the bit stream. [Figure 16-17](#) illustrates a simple matrix and a sequence. After decoding, data is rearranged from scanning sequence to linear sequence.

Then the inverse quantization is performed: each of the DCT coefficients is multiplied by a corresponding value from the weighting matrix `pQPMatrix` and by the quantizer scale factor, which are read from the bit stream.

This function is used in the MPEG-2 decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsH263VLCCodeErr</code>	Indicates an error when decoding in accordance with H263 standard.

## Inverse Quantization

Each of the decoded DCT coefficients in block should be inverse quantized through multiplication by the corresponding value from the weighting matrix and by the quantizer scale factor, which are read from the bit stream. Before this operation is performed, the DCT coefficients should be rearranged from zigzag scanning sequence to linear sequence.

For intra-frames all the DCT coefficients in block are processed, except the DC coefficient that should be processed separately in decoder, as described in MPEG standard. The functions for non-intra frames process all the DCT coefficients in block.

---

## QuantInvIntra\_MPEG2

*Performs inverse quantization for intra frames according to MPEG-2 standard.*

---

### Syntax

```
IppStatus ippiQuantInvIntra_MPEG2_16s_C1I(Ipp16s *pSrcDst, int QP, Ipp16s *pQPMatrix);
```

### Parameters

<i>pSrcDst</i>	Pointer to the start of the block.
<i>QP</i>	Quantizer scale factor read from the bit stream.
<i>pQPMatrix</i>	Pointer to the weighting matrix imposed by MPEG standard or user-defined.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiQuantInvIntra_MPEG2_16s_C1I` multiplies the DCT coefficients from *pSrcDst* by *QP* and by corresponding values from *pQPMatrix* and sends the result back to *pSrcDst*.

This function is used in the MPEG-2 encoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## QuantInv\_MPEG2

*Performs inverse quantization for non-intra frames according to MPEG-2 standard.*

---

### Syntax

```
IppStatus ippiQuantInv_MPEG2_16s_C1I(Ipp16s *pSrcDst, int QP, Ipp16s *pQPMatrix);
```

### Parameters

<i>pSrcDst</i>	Pointer to the start of the block.
<i>QP</i>	Quantizer read from the bit stream.
<i>pQPMatrix</i>	Pointer to the quantizing matrix imposed by MPEG standard or user-defined.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiQuantInv_MPEG2_16s_C1I` multiplies the DCT coefficients from *pSrcDst* by *QP* and by corresponding values from *pQPMatrix* and sends the result back to *pSrcDst*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## Inverse Discrete Cosine Transformation

---

### DCT8x8Inv\_AANTransposed\_16s\_C1R

*Performs inverse DCT on pre-transposed block.*

---

#### Syntax

```
ippiDCT8x8Inv_AANTransposed_16s_C1R(const Ipp16s* pSrc, Ipp16s* pDst, Ipp32s  
    dstStep, Ipp32s count);
```

#### Parameters

<i>pSrc</i>	Pointer to the block of DCT coefficients.
<i>pDst</i>	Pointer to the destination array.
<i>dstStep</i>	Step of the destination array.
<i>count</i>	Number of the last non-zero coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.

#### Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x8Inv_AANTransposed_16s_C1R` is used for non-intra macroblocks and performs inverse DCT on a transposed block. The block is transposed during the rearranging stage of VLC decoding, using the transposed scanning matrix as *scanMatrix* argument of the functions [ReconstructDCTBlock\\_MPEG1](#) and [ReconstructDCTBlock\\_MPEG2](#).

This function is used in the MPEG-2 decoder included into IPP Samples. See [introduction](#) to this section.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## DCT8x8Inv\_AANTransposed\_16s8u\_C1R

*Performs inverse DCT on pre-transposed block and converts output to unsigned char format.*

---

### Syntax

```
ippiDCT8x8Inv_AANTransposed_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, Ipp32s  
dstStep, Ipp32s count);
```

### Parameters

<i>pSrc</i>	Pointer to the block of DCT coefficients.
<i>pDst</i>	Pointer to the destination array.
<i>dstStep</i>	Step of the destination array.
<i>count</i>	Number of the last non-zero coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x8Inv_AANTransposed_16s8u_C1R` is used for intra macroblocks. The function performs inverse DCT on a transposed block and converts the output to unsigned char format. The block is transposed during the rearranging stage of VCL decoding, using the transposed scanning matrix as *scanMatrix* argument of the functions [ReconstructDCTBlockIntra MPEG1](#) and [ReconstructDCTBlockIntra MPEG2](#).

This function is used in the MPEG-2 decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## DCT8x8Inv\_AANTransposed\_16s\_P2C2R

*Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one array.*

---

### Syntax

```
ippiDCT8x8Inv_AANTransposed_16s_P2C2R(const Ipp16s* pSrcU, const Ipp16s* pSrcV,
    Ipp16s* pDstUV, Ipp32s dstStep, Ipp32s countU, Ipp32s countV);
```

### Parameters

<i>pSrcU</i>	Pointer to the block of DCT coefficients for <i>U</i> component.
<i>pSrcV</i>	Pointer to the block of DCT coefficients for <i>V</i> component.
<i>pDstUV</i>	Pointer to the destination array.
<i>dstStep</i>	Step of the destination array.
<i>countU</i>	Number of the last non-zero <i>U</i> coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.
<i>countV</i>	Number of the last non-zero <i>V</i> coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x8Inv_AANTransposed_16s_P2C2R` is used for non-intra macroblocks. The function performs inverse DCT on a transposed *U*-block and a transposed *V*-block and joins the output data into one [UV Block](#). The blocks are transposed during the rearranging stage of VCL decoding, using the transposed scanning matrix as *scanMatrix* argument of the functions [ReconstructDCTBlock\\_MPEG1](#) and [ReconstructDCTBlock\\_MPEG2](#).

This function is used in the MPEG-2 decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## DCT8x8Inv\_AANTransposed\_16s8u\_P2C2R

*Performs inverse DCT on pre-transposed data of two input chroma blocks and joins the output data into one unsigned char array.*

---

### Syntax

```
ippiDCT8x8Inv_AANTransposed_16s8u_P2C2R(const Ipp16s* pSrcU, const Ipp16s* pSrcV, Ipp8u* pDstUV, Ipp32s dstStep, Ipp32s countU, Ipp32s countV);
```

### Parameters

<i>pSrcU</i>	Pointer to the block of DCT coefficients for <i>U</i> component.
<i>pSrcV</i>	Pointer to the block of DCT coefficients for <i>V</i> component.
<i>pDstUV</i>	Pointer to the destination array.
<i>dstStep</i>	Step of the destination array.
<i>countU</i>	Number of the last non-zero <i>U</i> coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.
<i>countV</i>	Number of the last non-zero <i>V</i> coefficient in zig-zag order. If the block contains no non-zero coefficients, pass the value -1.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT8x8Inv_AANTransposed_16s8u_P2C2R` is used for intra macroblocks. The function performs inverse DCT on a transposed U-block and a transposed V-block and joins the output data into one unsigned char [UV Block](#). The blocks are transposed during the rearranging stage of VCL decoding, using the transposed scanning matrix as *scanMatrix* argument of the functions [ReconstructDCTBlockIntra\\_MPEG1](#) and [ReconstructDCTBlockIntra\\_MPEG2](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.



### Motion Compensation

This operation is applied to each block of non-intra type. The values of data obtained after inverse DCT are added to the data of predicted block and sent to the destination block. Prediction for half-pixel interpolation is performed by averaging of neighboring pixels with rounding. After processing, the data is converted from `Ipp16s` to `Ipp8u` type.

These functions are divided into two groups:

- Functions for predicted block (for block of P-type)
- Functions for bi-predicted block (for block of B-type).

See the detailed descriptions of these functions in [General Functions](#) section.

[Table 16-13](#) lists MC functions used in Video Data decoding.

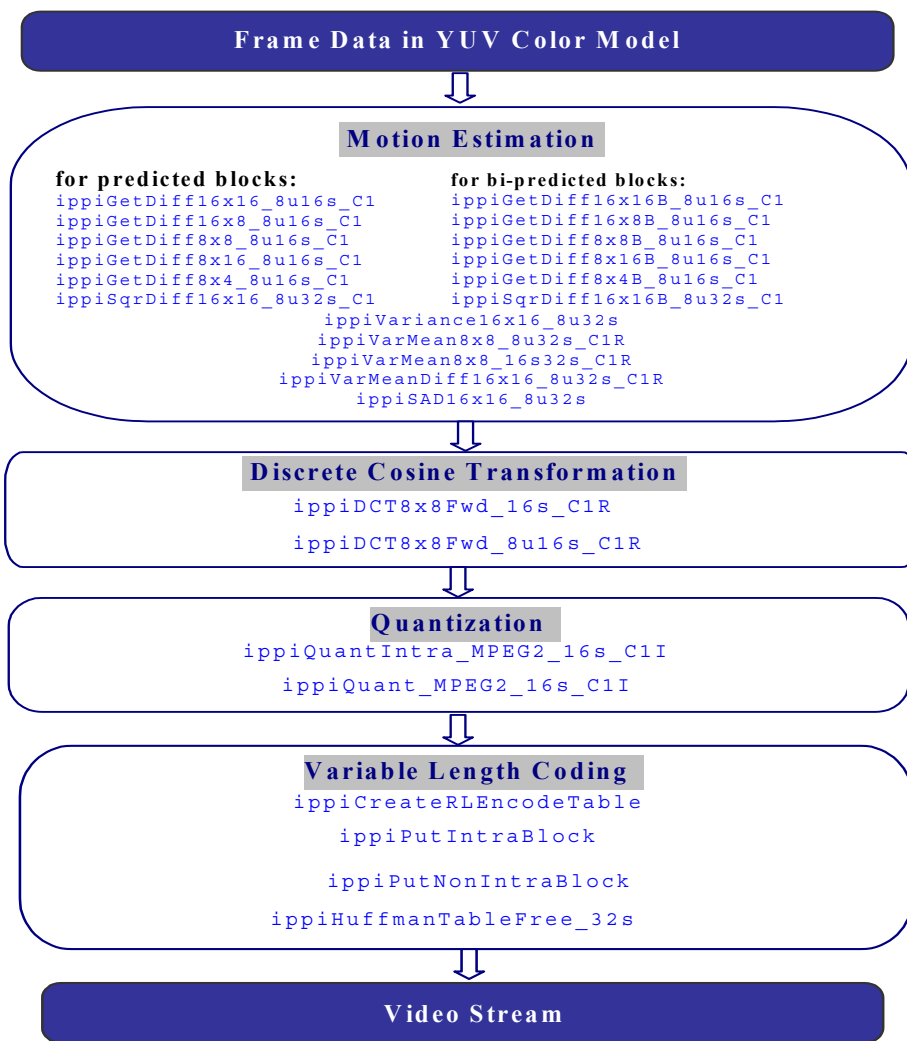
**Table 16-13     Motion Compensation Functions**

for predicted blocks:	for bi-predicted blocks:
<a href="#">MC16x4</a>	<a href="#">MC16x4B</a>
<a href="#">MC16x16</a>	<a href="#">MC16x16B</a>
<a href="#">MC16x8</a>	<a href="#">MC16x8B</a>
<a href="#">MC16x8UV</a>	<a href="#">MC16x8UVB</a>
<a href="#">MC8x16</a>	<a href="#">MC8x16B</a>
<a href="#">MC8x8</a>	<a href="#">MC8x8B</a>
<a href="#">MC8x4</a>	<a href="#">MC8x4B</a>

## Video Data Encoding

This section describes the main steps of MPEG video encoding according to the standard encoding pipeline shown in [Figure 16-18](#):

**Figure 16-18 Encoding Pipeline**



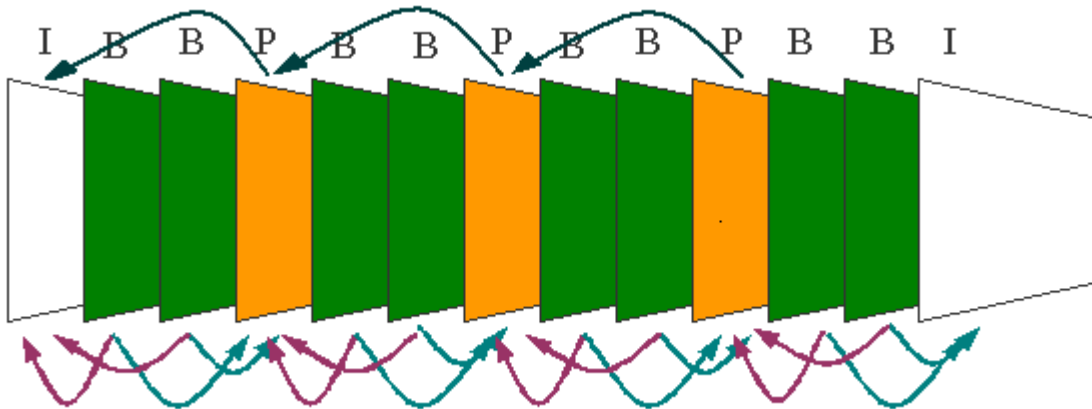
Note that for I and P frames after Quantization phase reconstruction may be implemented through inverse quantization ([QuantInv MPEG2](#)), inverse discrete cosine transformation (`ippiDCT8x8Inv_16s_C1R`, see [“DCT8x8Inv” on page 10-52](#)), and, for nonintra macroblocks, motion compensation in the same way as in the decoder (see [Figure 16-14](#)). The result is further used for motion estimation and compensation.

## Motion Estimation

The process of encoding starts with motion estimation (see [Figure 16-6](#) and [Figure 16-7](#)). Block *SrcRef* is found for each non-intra block *Dst* to be predicted, or two blocks *SrcRefF* and *SrcRefB* are found if block *Dst* is bi-predicted. The reference blocks are supposed to be similar to block *Dst*.

[Figure 16-19](#) shows how reference frames are selected.

**Figure 16-19 Intra, Predicted and Bi-Predicted Frames With Forward and Backward References**



The motion estimation process is performed by the following general Motion Estimation functions:

**Table 16-14     MPEG Video Motion Estimation Functions**

---

**Evaluation of difference between current predicted and reference blocks**

*For predicted blocks*

[GetDiff16x16](#)

[GetDiff16x8](#)

[GetDiff8x8](#)

[GetDiff8x16](#)

[GetDiff8x4](#)

*For bi-predicted blocks*

[GetDiff16x16B](#)

[GetDiff16x8B](#)

[GetDiff8x8B](#)

[GetDiff8x16B](#)

[GetDiff8x4B](#)

**Evaluation of sum of squares of differences between current and reference blocks**

*For predicted blocks*

[SqrDiff16x16](#)

*For bi-predicted blocks*

[SqrDiff16x16B](#)

**Evaluation of variance and mean of 8x8 block**

[VarMean8x8](#)

**Evaluation of variances and means of difference between two blocks**

[VarMeanDiff16x16](#)

[VarMeanDiff16x8](#)

**Evaluation of variance of current block**

[Variance16x16](#)

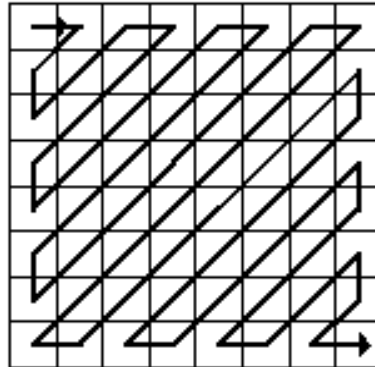
**Evaluation of sum of absolute difference between current and reference blocks**

[SAD16x16](#)

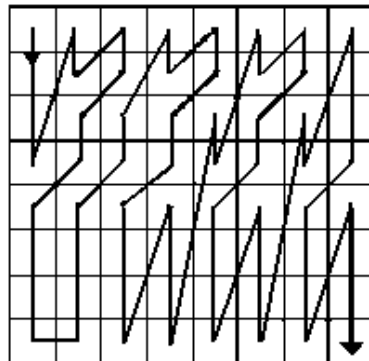
---

## Quantization

Quantization is applied to DCT coefficients to decrease their precision. You may use quatization matrices to vary precision coefficients with different positons. Scale factor is applied to all coefficients in the macroblock in the same way. Quantization values are often inverted to increase performance. Scan matrices specify the way to reorder 8x8 quantized DCT coefficients to an array of 64, so that statistically bigger values are located earlier; the further the element is from the first AC coefficient in the scanning sequence ([Figure 16-20](#)) the less important its value is for representing image and for further decoding.

**Figure 16-20 Default Scanning Sequence**

The alternate scanning sequence is used for interlaced video and provides better compression for field-coded blocks:

**Figure 16-21 Alternate Scanning Sequence**

---

## QuantIntra\_MPEG2

*Performs quantization on DCT coefficients for intra blocks in-place with specified quantization matrix according to MPEG-2 standard.*

---

### Syntax

```
IppStatus ippiQuantIntra_MPEG2_16s_C1I(Ipp16s *pSrcDst, Ipp32s QP, const Ipp32f *pQPMatrix, Ipp32s *pCount);
```

### Parameters

<i>pSrcDst</i>	Pointer to the block of DCT coefficients.
<i>QP</i>	Quantizer.
<i>pQPMatrix</i>	Pointer to the matrix of inverted quantization coefficients.
<i>pCount</i>	Number of the non-zero AC coefficients after quantization.

### Description

This function is declared in the `ippvc.h` header file. The in-place function `ippiQuantIntra_MPEG2_16s_C1I` multiplies all DCT coefficients in block except DC coefficients by the elements of the inverted quantization matrix and divide them by quantizer. The number of non-zero coefficients after quantization is stored in *pCount* for future considerations. If the pointer *pQPMatrix* is `NULL`, then the default matrix is used.

This function is used in the MPEG-2 encoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsDivByZeroErr</code>	Indicates an error when quantizer <i>QP</i> is equal to zero.

---

## Quant\_MPEG2

*Performs quantization on DCT coefficients for non-intra blocks in-place with specified quantization matrix according to MPEG-2 standard.*

---

### Syntax

```
IppStatus ippiQuant_MPEG2_16s_C1I(Ipp16s *pSrcDst, Ipp32s QP, const Ipp32f *pQPMatrix, Ipp32s *pCount);
```

### Parameters

<i>pSrcDst</i>	Pointer to the block of DCT coefficients.
<i>QP</i>	Quantizer.
<i>pQPMatrix</i>	Pointer to the matrix of inverted quantization coefficients.
<i>pCount</i>	Number of the non-zero coefficients after quantization.

### Description

This function is declared in the `ippvc.h` header file. The in-place function `ippiQuant_MPEG2_16s_C1I` multiplies DCT coefficients in block by the elements of the inverted quantization matrix and divide them by quantizer. The number of non-zero coefficients after quantization is stored in *pCount* for future considerations. If the pointer *pQPMatrix* is `NULL`, then the default matrix is used.

This function is used in the MPEG-2 encoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsDivByZeroErr</code>	Indicates an error when quantizer <i>QP</i> is equal to zero.

## Huffman Encoding Functions

After quantization the DCT coefficients in block should be encoded. Encoding is realized by Variable Length Coding (VLC), which uses standard code tables. These tables are entropy-constrained, that is, non-downloadable and optimized for a limited range of bit rates.

## VCL Table Building

---

## CreateRLEncodeTable

*Creates Run-Level Encode Table.*

---

### Syntax

```
IppStatus ippCreateRLEncodeTable (const Ipp32s *pSrcTable,  
    IppVCHuffmanSpec_32s** ppDstSpec);
```

### Parameters

<i>pSrcTable</i>	Pointer to the source table.
<i>ppDstSpec</i>	Double pointer to the destination encode table.

### Description

This function is declared in the `ippvc.h` header file. The function `ippCreateRLEncodeTable` creates the Run-Level Encode Table. The result is stored in block *ppDstSpec*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.



## Block Encoding Functions

### PutIntraBlock

*Encodes, rearranges and puts intra block into bit stream.*

#### Syntax

```
IppStatus ippPutIntraBlock(Ipp32u **ppBitStream, int *pOffset, Ipp16s*
    pSrcBlock, Ipp32s *pDCPred, IppVCHuffmanSpec_32u* pDCTable,
    IppVCHuffmanSpec_32s* pACTable, Ipp32s* pScanMatrix, Ipp32s EOBLen, Ipp32s
    EOBCode, Ipp32s count);
```

#### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bit stream.
<i>pOffset</i>	Pointer to the offset between the bit that <i>ppBitStream</i> points to and the start of the code.
<i>pSrcBlock</i>	Pointer to the block.
<i>pDCPred</i>	Pointer to the value to be added to the DC coefficient; this value is read from the special table imposed by standard.
<i>pDCTable</i>	Pointer to the table containing codes for DC coefficient, that is, the first coefficient among the DCT coefficients.
<i>pACTable</i>	Pointer to the table containing Run-Level codes for AC coefficients, that is, all DCT coefficients except the first coefficient.
<i>pScanMatrix</i>	Pointer to the scanning matrix.
<i>EOBLen</i>	Length of the block end code.
<i>EOBCode</i>	Value of the block end code.
<i>count</i>	Number of the non-zero AC coefficients.

#### Description

This function is declared in the `ippvc.h` header file. The function `ippPutIntraBlock` is applied to intra blocks and encodes a block of 8x8 DCT coefficients.

The `ippiPutIntraBlock` function encodes DC coefficient using the value `*pDCPred` for value computation and `pDCTable` for its encoding. Then it encodes AC coefficients using the `pACTable`.

The pointer `pScanMatrix` points to the scanning matrix that is described in MPEG standard. See [Figure 16-20](#) for the simple scanning matrix and sequence. After encoding, the data should be rearranged from linear sequence to scanning sequence.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## PutNonIntraBlock

*Encodes, rearranges and puts non-intra block into bit stream.*

---

### Syntax

```
IppStatus ippiPutNonIntraBlock(Ipp32u **ppBitStream, int* pOffset, Ipp16s*  
    pSrcBlock, IppVCHuffmanSpec_32s* pACTable, Ipp32s* pScanMatrix, Ipp32s  
    EOBLen, Ipp32s EOBCode, Ipp32s count);
```

### Parameters

<code>ppBitStream</code>	Double pointer to the current position in the bit stream.
<code>pOffset</code>	Pointer to the offset between the bit that <code>ppBitStream</code> points to and the start of the code.
<code>pSrcBlock</code>	Pointer to the block.
<code>pACTable</code>	Pointer to the table containing Run-Level codes for AC coefficients, that is, all DCT coefficients except the first coefficient.
<code>pScanMatrix</code>	Pointer to the scanning matrix.
<code>EOBLen</code>	Length of the block end code.
<code>EOBCode</code>	Value of the block end code.
<code>count</code>	Number of the non-zero coefficients.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiPutNonIntraBlock` is applied to predicted and bi-predicted non-intra blocks and encodes a block of 8x8 DCT coefficients. The function encodes both DC and AC coefficients using the *pACTable*.

The pointer *pScanMatrix* points to the scanning matrix that is described in MPEG standard. See [Figure 16-20](#) for the simple scanning matrix and sequence. After encoding, the data should be rearranged from linear sequence to scanning sequence.

## Return Values

<code>IppStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

## DV

DV is an international standard for a consumer digital video format.

DV uses a 1/4 inch (6.35mm) metal evaporate tape to record very high quality digital video. The video is sampled at the same rate as D-1, D-5, or Digital Betacam video – 720 pixels per scanline – although the color information is sampled at half the D-1 rate: 4:1:1 in 525-line (NTSC), and 4:2:0 in 625-line (PAL) formats.

DV uses intraframe compression: each compressed frame depends entirely on itself, and not on any data from preceding or following frames. However, it also uses adaptive *interfield* compression; if the compressor detects little difference between the two interlaced fields of a frame, it will compress them together, freeing up some of the "bit budget" to allow for higher overall quality. In theory, this means that static areas of images will be more accurately represented than areas with a lot of motion; in practice, this can sometimes be observed as a slight degree of "blockiness" in the immediate vicinity of moving objects.

The use of some functions described in this section is demonstrated of Intel® IPP Samples downloadable from

<http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm> .

The following section discusses the basic DV notions.

### DCT Block

The Y, U, V pixels in one frame shall be divided into *DCT blocks*. All DCT blocks for 625-50 system and DCT blocks except for rightmost DCT blocks in *U* and *V* for 525-60 system are structured with a rectangular area of eight vertical lines and eight horizontal pixels in a frame.

For 525-60 system, the rightmost DCT blocks in *U* and *V* are structured with 16 vertical lines and four horizontal pixels. The rightmost DCT block is reconstructed to eight vertical lines and eight horizontal pixels by moving the lower part of eight vertical lines and four horizontal pixels to the higher part of eight vertical lines and four horizontal pixels.

Figure 16-22 DCT Block Structure

		x								
y		0	1	2	3	4	5	6	7	
	0									Field 2
	1									Field 1
	2									Field 2
	3									Field 1
	4									Field 2
	5									Field 1
	6									Field 2
	7									Field 1

Six DCT blocks Y0, Y1, Y2, Y3, U, V form a *macroblock*.

Figure 16-23 Macroblock Structure for 525-60 System (Except Rightmost Macroblock)

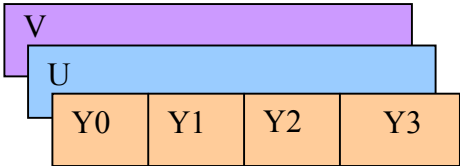
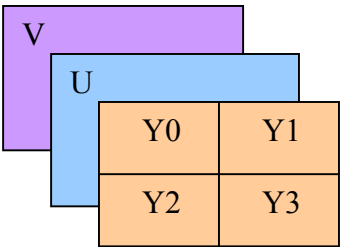


Figure 16-24 Macroblock Structure for 625-50 System and Rightmost Macroblock of 525-60 System



27 DCT macroblocks form a *superblock*. (Each cell stands for one macroblock).

Figure 16-25 DCT Superblock Structure for 525-60 System

0	11	12	23	24									8	9	20	21	0	11	12	23	24
1	10	13	22	25									7	10	19	22	1	10	13	22	
2	9	14	21	26									6	11	18	23	2	9	14	21	25
3	8	15	20										0	5	12	17	24	3	8	15	20
4	7	16	19										1	4	13	16	25	4	7	16	19
5	6	17	18										2	3	14	15	26	5	6	17	18

System 525-60 has three types of superblocks.

Figure 16-26     DCT Superblock Structure for 625-50 System

0	5	6	11	12	17	18	23	24
1	4	7	10	13	16	19	22	25
2	3	8	9	14	15	20	21	26

All superblocks of 625-50 system are of the same type.

A video *segment* consists of five macroblocks, which are gathered from various areas as below:

Figure 16-27     Macroblock Absolute Coordinates

Number of macroblock in the superblock	k	k	k	k	k
Superblock coordinates	$(2,(j+2)\%n)$	$(1,(j+6)\%n)$	$(3,(j+8)\%n)$	$(0,(j)\%n)$	$(4,(j+4)\%n)$

where  $k$  is within the interval  $[0,26]$ ,  $n = 10$  for 525-60 system,  $n = 12$  for 625-50 system, and  $j$  is the vertical order in the superblock.

Data in a video segment is compressed and transformed to the data of 385 bytes (*compressed segment*). A compressed segment consists of five compressed macroblocks. Each compressed macroblock consists of 77 bytes.

Figure 16-28     Arrangement of Compressed Macroblock

Byte sequence	H	cbY0	cbY1	cbY2	cbY3	cbU	cbV
Length	4	14	14	14	14	10	10

**Figure 16-29 Arrangement of Compressed Macroblock Header (H)**

---

Section name	Not used		qn0
Number of bits		4	
Number of bytes	3 bytes		1 byte

**Figure 16-30 Arrangement of Y compressed blocks cbY0, cbY1, cbY2, cbY3**

---

Section name	DC		m0	cl	AC	
Number of bits		1	1	2	4	
Number of bytes	1 byte	1 byte			12 bytes	

**Figure 16-31 Arrangement Cr compressed blocks cbU, cbV**

---

Section name	DC		m0	cl	AC	
Number of bits		1	1	2	4	
Number of bytes	1 byte	1 byte			8 bytes	

Type of block is equal to 1 when the difference between two fields is small (m0=0). Type of block is equal to 2 when the difference between two fields is big (m0=1).



Each DCT block is classified into four classes. For selecting quantization step, class number is used.

Table 16-15 DV Functions

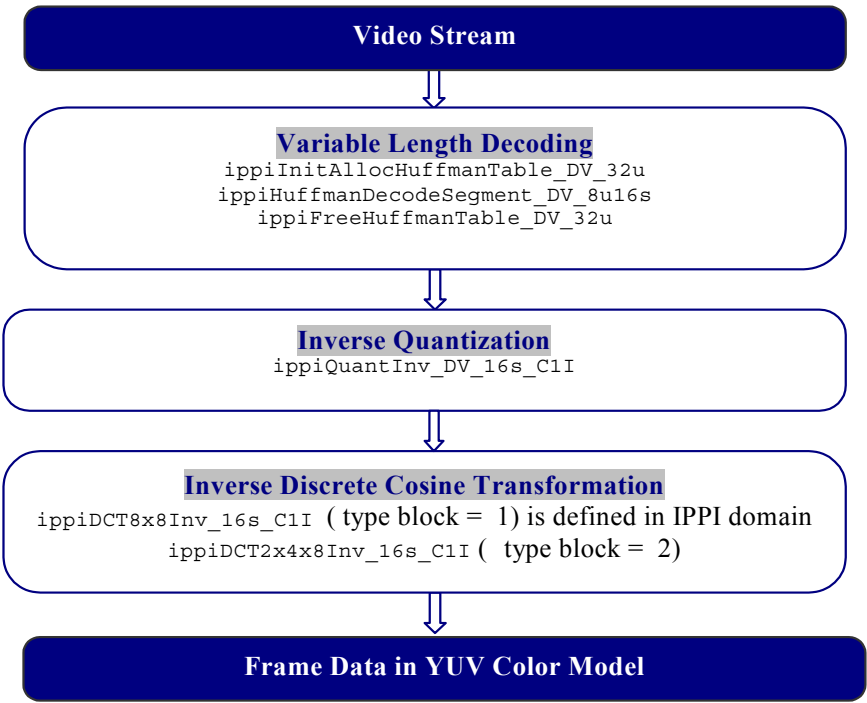
Function Short Name	Description
Decoding	
Variable Length Decoding	
<a href="#">InitAllocHuffmanTable_DV</a>	Allocates memory and initializes the table that contains codes for DCT coefficients (Run-Level codes).
<a href="#">HuffmanDecodeSegment_DV</a>	Decodes and rearranges segment block, multiplies first block element by 2.
<a href="#">FreeHuffmanTable_DV</a>	Frees the memory allocated for VLC table.
Inverse Quantization	
<a href="#">QuantInv_DV</a>	Performs inverse quantization on a block.
Inverse Discrete Cosine Transformation	
<a href="#">DCT2x4x8Inv</a>	Performs the inverse DCT for block of type 2 ( $m0=1$ ).
Encoding	
Discrete Cosine Transformation	
<a href="#">DCT2x4x8Frw</a>	Performs DCT for a block of type 2.
<a href="#">CountZeros8x8</a>	Evaluates number of zeros in a block.
Color Conversion	
<a href="#">YCrCb411ToYCbCr422_5MBDV,</a> <a href="#">YCrCb411ToYCbCr422_ZoomOut2_5MBDV,</a> <a href="#">YCrCb411ToYCbCr422_ZoomOut4_5MBDV,</a> <a href="#">YCrCb411ToYCbCr422_ZoomOut8_5MBDV</a>	Convert five macroblocks from YUV411 format into YUV2 format.
<a href="#">YCrCb411ToYCbCr422_EdgeDV,</a> <a href="#">YCrCb411ToYCbCr422_ZoomOut2_EdgeDV,</a> <a href="#">YCrCb411ToYCbCr422_ZoomOut4_EdgeDV,</a> <a href="#">YCrCb411ToYCbCr422_ZoomOut8_EdgeDV</a>	Convert a YCrCb411 macroblock into a YCrCb422 macroblock at the right edge of destination image.
<a href="#">YCrCb420ToYCbCr422_5MBDV,</a> <a href="#">YCrCb420ToYCbCr422_ZoomOut2_5MBDV,</a> <a href="#">YCrCb420ToYCbCr422_ZoomOut4_5MBDV,</a> <a href="#">YCrCb420ToYCbCr422_ZoomOut8_5MBDV</a>	Convert five YCrCb420 macroblocks into YCrCb422 macroblocks.

Table 16-15 DV Functions (continued)

Function Short Name	Description
<a href="#">YCrCb422ToYCbCr422_5MBDV</a> , <a href="#">YCrCb422ToYCbCr422_ZoomOut2_5MBDV</a> , <a href="#">YCrCb422ToYCbCr422_ZoomOut4_5MBDV</a> , <a href="#">YCrCb422ToYCbCr422_ZoomOut8_5MBDV</a>	Convert five YCrCb422 macroblocks into YCbCr422 macroblocks.

DV Decoding Functions

Figure 16-32 DV Decoding Pipeline



## Variable Length Decoding

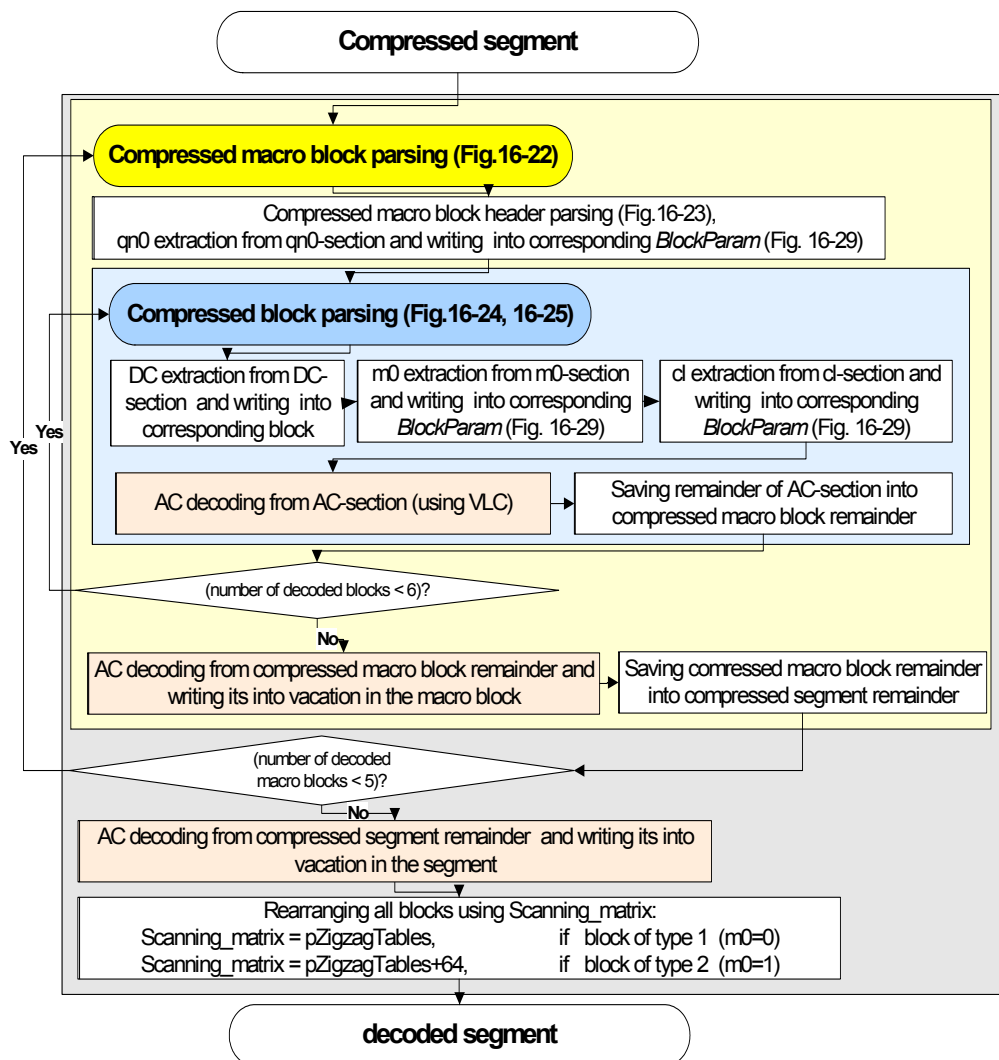
Structure of the source tables must be as follows:

### Example 16-4 Source Table Structure

---

static Ipp32s Table[]=	
{	
max_bits;	The maximum length of code
sub_sz1;	Not used
sub_sz2;	
N1;	The number of 1-bit codes
(code1, run1, level1);	The 1-bit codes, run values and level values.
(code2, run2, level2);	
...;	
(codeN1, runN1, levelN1);	
N2;	The number of 2-bit codes
(code1, run1, level1);	The 2-bit codes, run values and level values.
(code2, run2, level2);	
...;	
(codeN2, runN2, levelN2);	
...;	
Nm;	...
(code1, run1, level1);	The number of maximum length codes
(code2, run2, level2);	The maximum length codes, run values and level values.
...;	
(codeNm, runNm, levelNm);	
-1;	The significant value to indicate the end of table
};	

---

**Figure 16-33 Compressed Segment Decoding**


---

## InitAllocHuffmanTable\_DV

*Allocates memory and initializes table.*

---

### Syntax

```
IppStatus ippiInitAllocHuffmanTable_DV_32u(Ipp32s *pSrcTable1, Ipp32s  
*pSrcTable2, Ipp32u **ppHuffTable);
```

### Parameters

<i>pSrcTable1</i>	Pointer to Source Table 1.
<i>pSrcTable2</i>	Pointer to Source Table 2.
<i>ppHuffTable</i>	Double pointer to the destination decoding table.

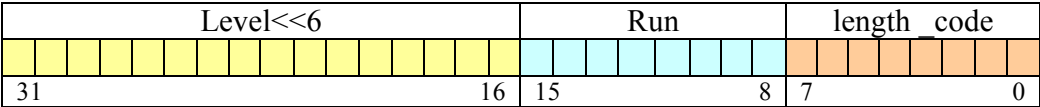
### Description

This function is declared in the `ippvc.h` header file. The function `ippiInitAllocHuffmanTable_DV_32u` allocates memory and initializes the table that contains codes for DCT coefficients (Run-Level codes). The function allocates memory at the address that *ppHuffTable* points to, and fills it with data from Source Table 1 that *pSrcTable1* points to and from Source Table 2 that *pSrcTable2* points to. See [Example 16-4](#) for the source table structure. When filling the destination decoding table the function multiplies the value of `level` by 64 (shift left by 6).

While decoding, the search for code is performed through the first subtable ([Figure 16-3](#)), which contains the most frequent and shortest codes. If the code value is not found, then the search continues through the following subtables, containing additional bits for longer codes. Those subtables also forward the search to the next ones, if the code value is not found.

This function is used in the DV decoder included into IPP Samples. See [introduction](#) to this section.

Figure 16-34 *HuFFT* Structure



length\_code = length code (if source tables == pSrcTable1),  
length\_code = length code + 4 (if source tables == pSrcTable2)

Return Values

- ippStsNoErr Indicates no error.
- ippStsNullPtrErr Indicates an error when at least one input pointer is NULL.
- ippStsMemAllocErr Indicates an error when no memory is allocated.

HuffmanDecodeSegment\_DV

Decodes and rearranges segment block, multiplies first block element by 128.

Syntax

```
IppStatus ippIHuffmanDecodeSegment_DV_8u16s(Ipp8u *pStream, Ipp32u *pZigzagTables, Ipp32u *pHuffTable, Ipp16s *pBlock, Ipp32u *pBlockParam);
```

Parameters

- pStream Pointer to bitstream (compressed segment).
- pZigzagTables Pointer to the array of two scanning matrices.
- pHuffTable Pointer to the decoding Huffman table.
- pBlock Pointer to decoded elements.

*pBlockParam*      Pointer to output parameters array [30].

Description

This function is declared in the `ippvc.h` header file. The function `ippiHuffmanDecodeSegment_DV` decodes and rearranges segment, multiplies the first block element by 128.

AC coefficients are decoded using Variable Length Codes with the help of the destination decoding table created by [InitAllocHuffmanTable\\_DV](#). Since this table contains `level` values multiplied by 64, the output values of AC coefficients are also multiplied by 64.

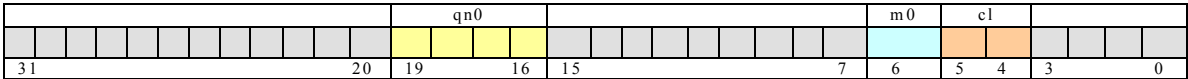
DC coefficients are obtained from the first 2 bytes of the compressed blocks (see [Figure 16-30](#) and [Figure 16-31](#)) with zeroing of the last 7 bits. The output values of DC coefficients are multiplied by 128 (shift left by 7).

The pointer *pStream* points to a compressed segment, which is a sequence of 5 compressed macroblocks.

Scanning\_matrix = *pZigzagTables*, if the block is of type 1 (`m0 = 0`). Scanning\_matrix = *pZigzagTables* + 64, if the block is of type 2 (`m0 = 1`).

The argument *\*pBlockParam* ([Figure 16-35](#)) stores values `m0`, `c1` for every block. Value `qn0` is obtained from the first block of the macroblock. The same value `qn0` is used for all blocks of the macroblock.

Figure 16-35    *BlockParam* Structure



This function is used in the DV decoder included into IPP Samples. See [introduction](#) to this section.

Return Values

- `ippStsNoErr`              Indicates no error.
- `ippStsNullPtrErr`       Indicates an error when at least one input pointer is NULL.

---

## FreeHuffmanTable\_DV

*Frees the memory allocated for VLC table.*

---

### Syntax

```
IppStatus ippiFreeHuffmanTable_DV_32u(Ipp32u *pHuffTable);
```

### Parameters

*pHuffTable*      Pointer to the decoding table.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiFreeHuffmanTable_DV_32u` frees the memory at *pHuffTable* allocated for VLC table.

This function is used in the DV decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

`ippStsNoErr`              Indicates no error.  
`ippStsNullPtrErr`      Indicates an error when *pHuffTable* is NULL.

### Inverse Quantization

---

## QuantInv\_DV

*Performs inverse quantization on a block.*

---

### Syntax

```
IppStatus ippiQuantInv_DV_16s_C1I(Ipp16s *pSrcDst, Ipp16s *pDequantTable);
```

### Parameters

*pSrcDst*              Pointer to the block.



*pDequantTable* Pointer to the dequantization table.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiQuantInv_DV_16s_C1I` performs inverse quantization on a block. Each of the decoded DCT coefficients in the block should be inverse quantized through multiplying by a corresponding value from the weighting matrix and division on  $2^{14}$ . Selection dequantization table depends of the block type.

This function is used in the DV decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## Inverse Discrete Cosine Transformation

---

### DCT2x4x8Inv

*Performs the inverse DCT for block of type 2 ( $m0=1$ ).*

---

## Syntax

```
IppStatus ippiDCT2x4x8Inv_16s_C1I(Ipp16s *pSrcDst);
```

## Parameters

*pSrcDst* Pointer to the block.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiDCT2x4x8Inv_16s_C1I` performs the inverse DCT for a block of type 2 ( $m0 = 1$ ). Formula for inverse DCT is as follows:

$$P(x, 2^*z) = \sum_{u=0}^3 \sum_{h=0}^7 (c(u)c(h)(c(h, u) + c(h, u + 4))KC)$$

$$P(x, 2^*z + 1) = \sum_{u=0}^3 \sum_{h=0}^7 (c(u)c(h)(c(h, u) - c(h, u + 4))KC) ,$$

where

$x$  in  $[0, 7]$

$y$  in  $[0, 7]$

$z = \text{int}(\frac{Y}{2})$

$$KC = \cos\left(\frac{\pi u(2z + 1)}{8}\right) \cos\left(\frac{\pi h(2x + 1)}{16}\right)$$

$$c(h) = \frac{0.5}{\text{sqr}(2)} \quad \text{for } h = 0,$$

$$c(h) = 0.5 \quad \text{for } h \text{ in } [1, 7],$$

$$c(u) = \frac{0.5}{\text{sqr}(2)} \quad \text{for } u = 0,$$

$$c(u) = 0.5 \quad \text{for } u \text{ in } [1, 7]$$

This function is used in the DV decoder included into IPP Samples. See [introduction](#) to this section.

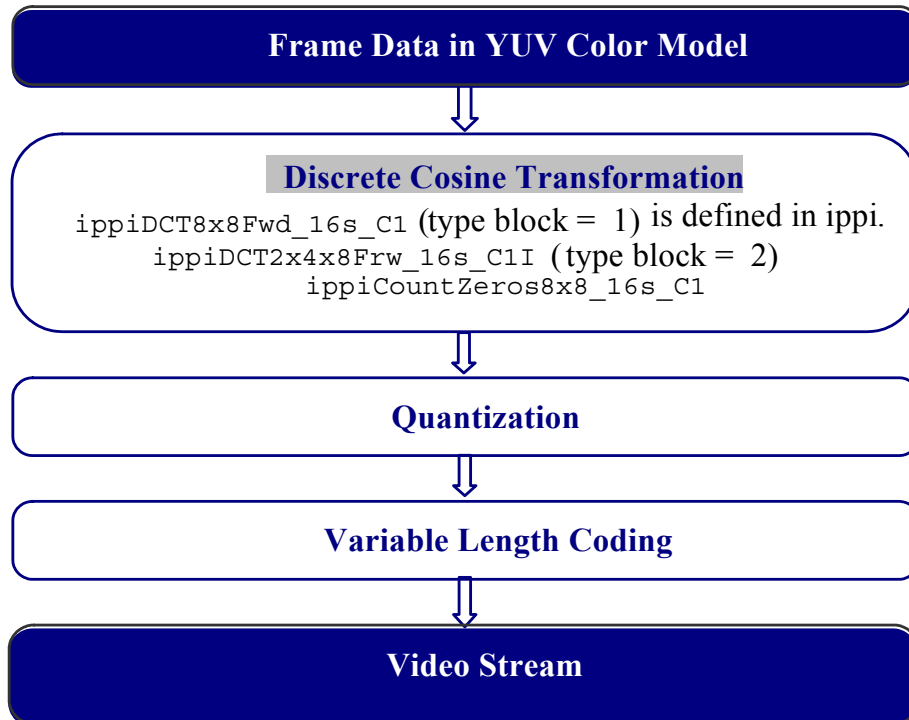
### Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error when `pSrcDst` is NULL.

## DV Encoding Functions

Figure 16-36 DV Encoding Pipeline



## Discrete Cosine Transformation

---

### DCT2x4x8Frw

*Performs DCT for a block of type 2.*

---

#### Syntax

```
IppStatus ippIDCT2x4x8Frw_16s_C1I(Ipp16s *pSrcDst);
```

#### Parameters

*pSrcDst*                      Pointer to the block.

#### Description

This function is declared in the `ippvc.h` header file. The function `ippIDCT2x4x8Frw_16s_C1I` performs the DCT for block of type 2. Formula for DCT is as follows:

$$C(h, u) = c(u) c(h) \sum_{z=0}^3 \sum_{x=0}^7 ((P(x, 2z) + P(x, 2z + 1)) KC)$$

$$C(h, u + 4) = c(u) c(h) \sum_{z=0}^3 \sum_{x=0}^7 ((P(x, 2z) - P(x, 2z + 1)) KC) ,$$

where

$h$  in  $[0, 7]$

$u$  in  $[0, 7]$

$z = \text{int}(\frac{Y}{2})$

$$KC = \cos\left(\frac{\pi u(2z + 1)}{8}\right) \cos\left(\frac{\pi h(2x + 1)}{16}\right)$$

$$c(h) = \frac{0.5}{\text{sqr}(2)} \quad \text{for } h = 0,$$

$$c(h) = 0.5 \quad \text{for } h \text{ in } [1, 7],$$

$$c(u) = \frac{0.5}{\text{sqr}(2)} \quad \text{for } u = 0,$$

$$c(u) = 0.5 \quad \text{for } u \text{ in } [1, 7].$$

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when <i>pSrcDst</i> is NULL.

---

## CountZeros8x8

*Evaluates number of zeros in a block.*

---

### Syntax

```
IppStatus ippiCountZeros8x8_16s_C1(Ipp16s *pSrc, Ipp32u* pCount);
```

### Parameters

<i>pSrc</i>	Pointer to the block.
<i>pCount</i>	Pointer to the number of zeros.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiCountZeros8x8_16s_C1` is used to identify the type of a block by comparing the results from DCT of the first type and DCT of the second type performed on the block.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is NULL.

## DV Color Conversion Functions

These functions convert color format in DV video processing.

The minimal coding unit in DV is a set of five macroblocks, with each macroblock comprising four Y-blocks, one Cb block, and one Cr block. The macroblocks are taken from different parts of the processed frame to decrease loss of quality in case the film is damaged. The data format for a DV frame is

- YUV411 in system 525, or NTSC,
- interlaced YUV420 in system 625, or PAL.

These formats are converted into a more popular YUY2 format after inverse discrete cosine transformation.

The functions described in this subsection are used in the DV decoder included into IPP Samples. See [introduction](#) to DV section.

---

### **YCrCb411ToYCbCr422\_5MBDV, YCrCb411ToYCbCr422\_ZoomOut2\_5MBDV, YCrCb411ToYCbCr422\_ZoomOut4\_5MBDV, YCrCb411ToYCbCr422\_ZoomOut8\_5MBDV**

*Convert five YCrCb411 macroblocks into YCrCb422 macroblocks.*

---

#### **Syntax**

```
IppStatus ippiYCrCb411ToYCbCr422_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5],  
    Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R(const Ipp16s*  
    pSrc[5], Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R(const Ipp16s*  
    pSrc[5], Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R(const Ipp16s*  
    pSrc[5], Ipp8u* pDst[5]);
```

## Parameters

<i>pSrc</i>	Array of pointers to the five source macroblocks.
<i>pDst</i>	Array of pointers to the five destination macroblocks.
<i>dstStep</i>	Step for the destination image.

## Description

These functions convert YCrCb411 macroblocks to YCbCr422 macroblocks, that is, they transfer the five macroblocks of uncompressed DV data after IDCT processing from the internal format YUV411 to the destination buffer in YUV2 format with saturation.

The function `ippiYCrCb411ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R` also reduces the size of the destination image by 2 times, `ippiYCrCb411ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R` - by 4 times, and `ippiYCrCb411ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R` - by 8 times.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is NULL.

---

## YCrCb411ToYCbCr422\_EdgeDV, YCrCb411ToYCbCr422\_ZoomOut2\_EdgeDV, YCrCb411ToYCbCr422\_ZoomOut4\_EdgeDV, YCrCb411ToYCbCr422\_ZoomOut8\_EdgeDV

*Convert a YCrCb411 macroblock into a YCrCb422 macroblock at the right edge of destination image.*

---

## Syntax

```

IppStatus ippiYCrCb411ToYCbCr422_EdgeDV_16s8u_P3C2R(const Ipp16s* pSrc, Ipp8u*
    pDst, int dstStep);

IppStatus ippiYCrCb411ToYCbCr422_ZoomOut2_EdgeDV_16s8u_P3C2R(const Ipp16s*
    pSrc, Ipp8u* pDst, int dstStep);

IppStatus ippiYCrCb411ToYCbCr422_ZoomOut4_EdgeDV_16s8u_P3C2R(const Ipp16s*
    pSrc, Ipp8u* pDst, int dstStep);

```

```
IppStatus ippiYCrCb411ToYCbCr422_ZoomOut8_EdgeDV_16s8u_P3C2R(const Ipp16s*
    pSrc, Ipp8u* pDst, int dstStep);
```

## Parameters

<i>pSrc</i>	Pointer to the source macroblock.
<i>pDst</i>	Pointer to the destination macroblock.
<i>dstStep</i>	Step for the destination image.

## Description

These functions convert a YCrCb411 macroblock into a YCrCb422 macroblock at the right edge of destination image. These functions are intended for NTSC standard conversion as all right edge macroblocks in NTSC have slightly different locations.

The function `ippiYCrCb411ToYCbCr422_ZoomOut2_EdgeDV_16s8u_P3C2R` also reduces the size of the destination image by 2 times,

`ippiYCrCb411ToYCbCr422_ZoomOut4_EdgeDV_16s8u_P3C2R` - by 4 times, and  
`ippiYCrCb411ToYCbCr422_ZoomOut8_EdgeDV_16s8u_P3C2R` - by 8 times.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is NULL.

---

## YCrCb420ToYCbCr422\_5MBDV, YCrCb420ToYCbCr422\_ZoomOut2\_5MBDV, YCrCb420ToYCbCr422\_ZoomOut4\_5MBDV, YCrCb420ToYCbCr422\_ZoomOut8\_5MBDV

*Convert five YCrCb420 macroblocks into YCrCb422 macroblocks.*

---

## Syntax

```
IppStatus ippiYCrCb420ToYCbCr422_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5],
    Ipp8u* pDst[5], int dstStep);
```



```
IppStatus ippiYCrCb420ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R(const Ipp16s*  
    pSrc[5], Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb420ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R(const Ipp16s*  
    pSrc[5], Ipp8u* pDst[5], int dstStep);  
IppStatus ippiYCrCb420ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R(const Ipp16s*  
    pSrc[5], Ipp8u* pDst[5], int dstStep);
```

### Parameters

<i>pSrc</i>	Array of pointers to the five source macroblocks.
<i>pDst</i>	Array of pointers to the five destination macroblocks.
<i>dstStep</i>	Step for the destination image.

### Description

These functions convert YCrCb420 macroblocks to YCbCr422 macroblocks, that is, they transfer the five macroblocks of uncompressed DV data after IDCT processing from the internal format YUV420 (PAL format) to the destination buffer in YUV2 format with saturation.

The function `ippiYCrCb420ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R` also reduces the size of the destination image by 2 times,

`ippiYCrCb420ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R` - by 4 times, and

`ippiYCrCb420ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R` - by 8 times.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one of the pointers is NULL.

## YCrCb422ToYCbCr422\_5MBDV, YCrCb422ToYCbCr422\_ZoomOut2\_5MBDV, YCrCb422ToYCbCr422\_ZoomOut4\_5MBDV, YCrCb422ToYCbCr422\_ZoomOut8\_5MBDV

*Convert five YCrCb422 macroblocks into YCrCb422 macroblocks.*

---

### Syntax

```
IppStatus ippiYCrCb422ToYCbCr422_5MBDV_16s8u_P3C2R(const Ipp16s* pSrc[5],
    Ipp8u* pDst[5], int dstStep);
IppStatus ippiYCrCb422ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R(const Ipp16s*
    pSrc[5], Ipp8u* pDst[5], int dstStep);
IppStatus ippiYCrCb422ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R(const Ipp16s*
    pSrc[5], Ipp8u* pDst[5], int dstStep);
IppStatus ippiYCrCb422ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R(const Ipp16s*
    pSrc[5], Ipp8u* pDst[5]);
```

### Parameters

<i>pSrc</i>	Array of pointers to the five source macroblocks.
<i>pDst</i>	Array of pointers to the five destination macroblocks.
<i>dstStep</i>	Step for the destination image.

### Description

These functions convert YCrCb422 macroblocks to YCbCr422 macroblocks. These functions may be used in dv50 decoders.

The function `ippiYCrCb422ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R` also reduces the size of the destination image by 2 times,  
`ippiYCrCb422ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R` - by 4 times, and  
`ippiYCrCb422ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R` - by 8 times.

Note a specific feature of decoding in this case. In spite of the fact that the input data format is YUV422, the *pSrc* pointer should point to the macroblocks that contain the six blocks in the following order:

[Y0 block] [empty block] [Y1 block] [empty block] [V block] [U block] .

### Return Values

`ippStsNoErr`            Indicates no error.

`ippStsNullPtrErr`    Indicates an error when at least one of the pointers is NULL.

## MPEG-4

This section contains ippVC functions for encoding and decoding of video data according to ISO/IEC 14496-2 MPEG-4 standard (see [[ISO14496A](#)]).

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm>.

**Table 16-16 MPEG-4 Video Decoder Functions**

Function Short Name	Description
<b>Motion Compensation</b>	
<a href="#">Copy8x8QP MPEG4</a> , <a href="#">Copy16x8QP MPEG4</a> , <a href="#">Copy16x16QP MPEG4</a>	Copy a block with quarter-pixel accuracy.
<a href="#">OBMC8x8HP MPEG4</a> , <a href="#">OBMC16x16HP MPEG4</a> , <a href="#">OBMC8x8QP MPEG4</a> ,	Perform the overlapped block motion compensation (OBMC).
<b>Sprite and Global Motion Compensation</b>	
<a href="#">WarpInit MPEG4</a>	Initializes IppiWarpSpec_MPEG4 structure for further usage in GMC or Sprite reconstruction.
<a href="#">WarpGetSize MPEG4</a>	Returns size of IppiWarpSpec_MPEG4 structure.
<a href="#">WarpLuma MPEG4</a>	Warps arbitrary rectangular luminance region.
<a href="#">WarpChroma MPEG4</a>	Warps arbitrary rectangular chrominance region.
<a href="#">CalcGlobalMV MPEG4</a>	Calculates Global Motion Vector for one macroblock.
<a href="#">ChangeSpriteBrightness MPEG4</a>	Change brightness after sprite warping.
<b>Motion Vector Decoding and Padding</b>	
<a href="#">DecodePadMV PVOP MPEG4</a>	Decodes and pads four motion vectors of the non-intra macroblock in P-VOP.
<a href="#">DecodeMV BVOP Forward MPEG4</a>	Decodes motion vector of the macroblock in B-VOP forward mode.
<a href="#">DecodeMV BVOP Backward MPEG4</a>	Decodes motion vector of the macroblock in B-VOP backward mode.
<a href="#">DecodeMV BVOP Interpolate MPEG4</a>	Decodes motion vector of the macroblock in B-VOP interpolate mode.
<a href="#">DecodeMV BVOP Direct MPEG4</a>	Decodes motion vectors of the macroblock in B-VOP using direct mode.

**Table 16-16 MPEG-4 Video Decoder Functions (continued)**

Function Short Name	Description
<a href="#"><u>DecodeMV_BVOP_DirectSkip_MPEG4</u></a>	Decodes motion vectors of the macroblock in B-VOP using direct mode when the current macroblock is skipped.
<a href="#"><u>LimitMVToRect_MPEG4</u></a>	Limits the motion vector of current block/macroblock into the extended bounding rectangle.
<b>Coefficient Prediction and Reconstruction</b>	
<a href="#"><u>PredictReconCoefIntra_MPEG4</u></a>	Performs adaptive DC/AC coefficient prediction for an intra block.
<b>Motion Padding</b>	
<a href="#"><u>PadMBHorizontal_MPEG4</u></a>	Performs horizontal extended padding process on exterior macroblock immediately next to boundary macroblocks.
<a href="#"><u>PadMBVertical_MPEG4</u></a>	Performs vertical extended padding process on exterior macroblock immediately next to boundary macroblocks.
<a href="#"><u>PadMBGray_MPEG4</u></a>	Fills gray value in exterior macroblock that is not located next to any boundary macroblocks.
<a href="#"><u>PadCurrent_16x16_MPEG4,</u></a> <a href="#"><u>PadCurrent_8x8_MPEG4</u></a>	Perform horizontal and vertical repetitive padding process on luminance/alpha macroblock or chrominance block.
<b>Vector Padding</b>	
<a href="#"><u>PadMV_MPEG4</u></a>	Performs vector padding for a non-transparent macroblock.
<b>Inverse Quantization</b>	
<a href="#"><u>QuantInvIntraInit_MPEG4,</u></a> <a href="#"><u>QuantInvInterInit_MPEG4</u></a>	Initialize specification structures for QuantInvIntra_MPEG4 and QuantInvInter_MPEG4 respectively.
<a href="#"><u>QuantInvIntraGetSize_MPEG4,</u></a> <a href="#"><u>QuantInvInterGetSize_MPEG4</u></a>	Return the size of the initialized specification structures.
<a href="#"><u>QuantInvIntra_MPEG4,</u></a> <a href="#"><u>QuantInvInter_MPEG4</u></a>	Perform inverse quantization on intra/inter coded block.
<b>VLC Decoding</b>	
<a href="#"><u>DecodeDCIntra_MPEG4</u></a>	Decodes one DC coefficient for intra coded block.
<a href="#"><u>DecodeCoeffsIntra_MPEG4</u></a>	Decodes DCT coefficients for intra coded block.

**Table 16-16 MPEG-4 Video Decoder Functions (continued)**

Function Short Name	Description
<a href="#"><u>DecodeCoeffsIntraRVLCBack MPEG4</u></a>	Decodes DCT coefficients in backward direction for intra coded block using RVLC.
<a href="#"><u>DecodeCoeffsInter MPEG4</u></a>	Decodes DCT coefficients for inter coded block.
<a href="#"><u>DecodeCoeffsInterRVLCBack MPEG4</u></a>	Decodes DCT coefficients in backward direction for inter coded block using RVLC.
<a href="#"><u>ReconstructCoeffsInter MPEG4</u></a>	Decodes DCT coefficients, performs inverse scan and inverse quantization for inter coded block.
<a href="#"><u>DecodeVLCZigzag IntraDCVLC MPEG4,</u></a> <a href="#"><u>DecodeVLCZigzag IntraACVLC MPEG4</u></a>	Perform VLC decoding and inverse zigzag scan for one intra coded block.
<a href="#"><u>DecodeVLCZigzag Inter MPEG4</u></a>	Performs VLC decoding and inverse zigzag scan for one inter coded block.
<b>Block Decoding</b>	
<a href="#"><u>DecodeBlockCoef Intra MPEG4</u></a>	Decodes the INTRA block coefficients.
<a href="#"><u>DecodeBlockCoef Inter MPEG4</u></a>	Decodes the INTER block coefficients.
<b>Postprocessing</b>	
<a href="#"><u>FilterDeblocking8x8HorEdge MPEG4,</u></a> <a href="#"><u>FilterDeblocking8x8VerEdge MPEG4</u></a>	Perform deblocking filtering on a horizontal or vertical edge of two adjacent blocks.
<a href="#"><u>FilterDeringingThreshold MPEG4</u></a>	Computes threshold values for the deringing filtering through a macroblock.
<a href="#"><u>FilterDeringingSmooth8x8 MPEG4</u></a>	Performs deringing filtering of a block.
<b>Shape Decoding</b>	
<a href="#"><u>DecodeCAEIntraH MPEG4,</u></a> <a href="#"><u>DecodeCAEIntraV MPEG4</u></a>	Perform Context Arithmetic Code decoding in the intra macroblock.
<a href="#"><u>DecodeCAEInterH MPEG4,</u></a> <a href="#"><u>DecodeCAEInterV MPEG4</u></a>	Perform Context Arithmetic Code decoding in the inter macroblock.
<a href="#"><u>DecodeMVS MPEG4</u></a>	Decodes motion vectors of shape according to specification.
<a href="#"><u>PadMBPartial MPEG4</u></a>	Performs general padding if the current macroblock is partial.
<a href="#"><u>PadMBTransparent MPEG4</u></a>	Performs general padding if the current macroblock is transparent.
<a href="#"><u>PadMBOpaque MPEG4</u></a>	Performs general padding if the current macroblock is opaque.

**Table 16-17    MPEG-4 Video Encoder Functions**

Function Short Name	Description
<b>Motion Estimation</b>	
<a href="#"><u>SumNorm VOP MPEG4</u></a>	Performs summation of a block of indicated size.
<a href="#"><u>BlockMatch Integer 16x16 SEA</u></a>	Performs 16x16 size block match with sub-region.
<a href="#"><u>MotionEstimation 16x16 SEA</u></a>	Completes 16X16 size motion estimation using core SEA.
<a href="#"><u>BlockMatch Integer 16x16 MVFAST</u></a>	Performs 16x16 size block match with large and/or small diamond search.
<a href="#"><u>MotionEstimation 16x16 MVFAST</u></a>	Performs fast motion estimation using the MVFAST algorithm.
<a href="#"><u>ComputeTextureErrorBlock SAD</u></a>	Computes texture error of the block with SAD exported.
<a href="#"><u>ComputeTextureErrorBlock</u></a>	Computes texture error of the block.
<b>Quantization</b>	
<a href="#"><u>QuantIntraInit MPEG4,</u></a> <a href="#"><u>QuantInterInit MPEG4</u></a>	Initialize specification structures for QuantIntra_MPEG4 and QuantInter_MPEG4 respectively.
<a href="#"><u>QuantIntraGetSize MPEG4,</u></a> <a href="#"><u>QuantInterGetSize MPEG4</u></a>	Return the size of the initialized specification structures.
<a href="#"><u>QuantIntra MPEG4, QuantInter MPEG4</u></a>	Perform quantization on intra/inter coded block.
<b>VLC Encoding</b>	
<a href="#"><u>EncodeDCIntra MPEG4</u></a>	Encodes one DC coefficient for intra coded block.
<a href="#"><u>EncodeCoeffsIntra MPEG4</u></a>	Encodes DCT coefficients for intra coded block.
<a href="#"><u>EncodeCoeffsInter MPEG4</u></a>	Encodes DCT coefficients for inter coded block.
<a href="#"><u>EncodeVLCZigzag IntraDCVLC MPEG4,</u></a> <a href="#"><u>EncodeVLCZigzag IntraACVLC MPEG4</u></a>	Perform zigzag scanning and VLC encoding for one intra block.
<a href="#"><u>EncodeVLCZigzag Inter MPEG4</u></a>	Performs classical zigzag scanning and VLC encoding for one inter block.
<b>Block Encoding</b>	
<a href="#"><u>TransRecBlockCoef inter MPEG4</u></a>	Implements DCT, quantizes DCT coefficients of the inter block, and reconstructs the texture residual in the process.

**Table 16-17 MPEG-4 Video Encoder Functions (continued)**

Function Short Name	Description
<a href="#"><u>TransRecBlockCoef_intra MPEG4</u></a>	Quantizes DCT coefficients, implements AC/DC coefficients prediction of the intra block, and stores them into buffer.
<b>MV Encoding</b>	
<a href="#"><u>FindMVpred MPEG4</u></a>	Finds the vector predictor from three candidates.
<a href="#"><u>EncodeMV MPEG4</u></a>	Finds the prediction MV and encodes the difference.

## MPEG-4 Video Decoder Functions

This section describes Intel IPP functions that are built to support the ISO/IEC 14496-2 MPEG-4 video decoder. MPEG-4 (see [[ISO14496](#)]) is a widely used coding method for video signals in various applications such as digital storage media, Internet, various forms of wired or wireless communications, etc.

The functions cover the following aspects of MPEG-4 decoder:

- progressive, non-scalable texture decoding and shape (grayscale) decoding
- macroblock-based repetitive and extended padding
- block-based variable length code (VLC) decoding and inverse zigzag scan
- motion vector decoding and padding



- intra DC/AC prediction
- overlapped block motion compensation
- block layer coefficient decoding, including bitstream parsing, VLC decoding, intra DC/AC prediction (for intra blocks), inverse quantization, inverse zigzag and IDCT, with appropriate clipping on each step
- motion compensation
- postprocessing for coding noise reduction including deblocking and deringing filters.



---

**NOTE.** All MPEG-4 functions can be applied if `short_video_header` equals 0. If `short_video_header` equals 1, use H.263 decoder functions.

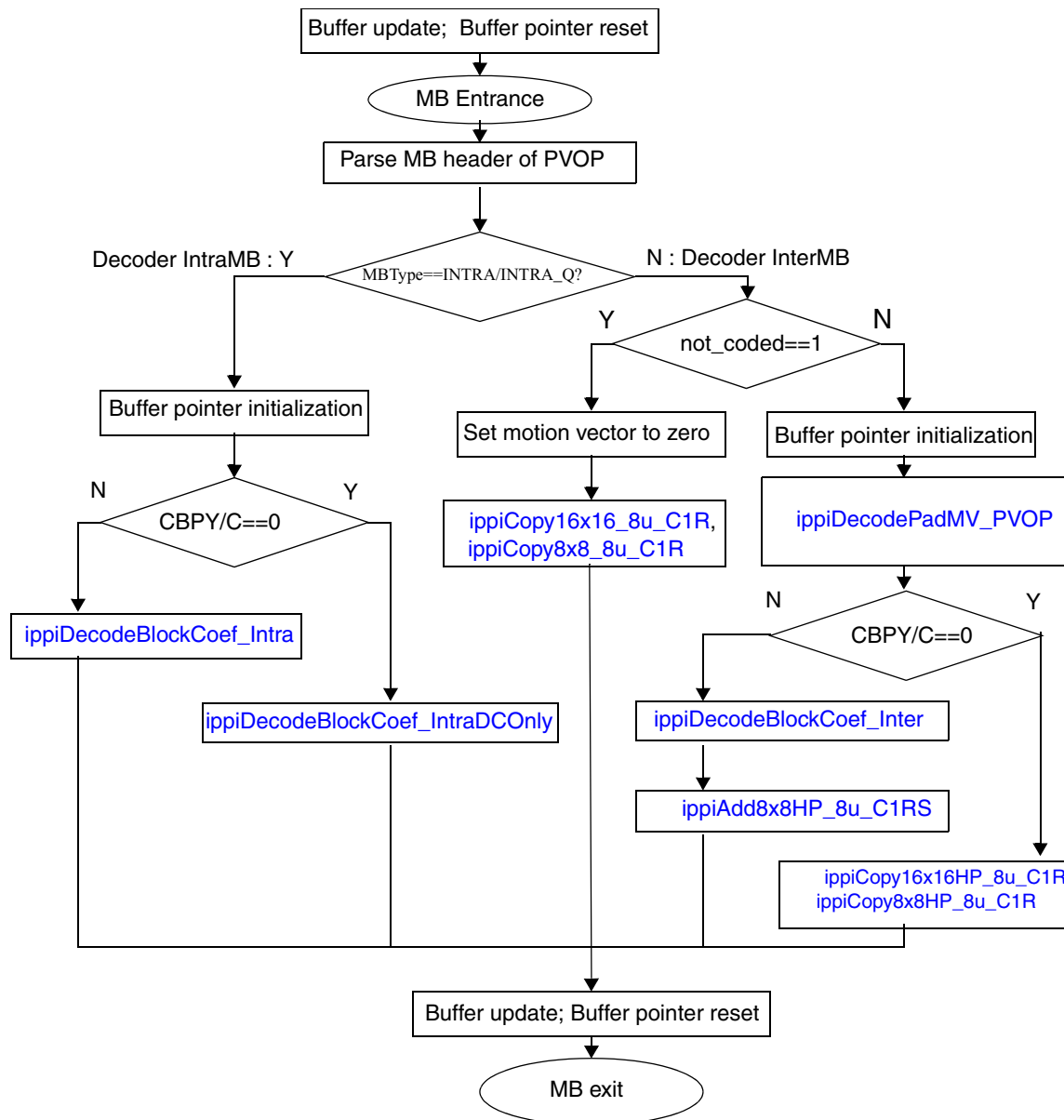
---

Below a high-level description of the MPEG-4 functions usage is given, followed by Intel IPP macro/data structures definition and the detailed descriptions of individual functions.

## High Level Description

[Figure 16-37](#) shows the general steps that are needed to decode a MacroBlock (MB) in Predictive coded Video Object Plane (P-VOP) and the associated Intel IPP functions.

Figure 16-37 Flowchart of Decoding a MB in P-VOP



## Data Types and Structures

### Video Components

Video components are defined as follows:

```
typedef enum {  
    IPP_VIDEO_LUMINANCE,      /* Luminance component */  
    IPP_VIDEO_CHROMINANCE,    /* Chrominance component */  
    IPP_VIDEO_ALPHA           /* Alpha component */  
} IppVideoComponent;
```

### Pixel Planes and Alpha Plane

The decoder output is stored in three pixel planes denoted as Y plane (luminance component), Cb plane and Cr plane (chrominance components), plus one A plane (alpha signal).

The size of Y plane relates to, but is not equal to, that of the Video Object Layer (VOL) as a result of the VOP expansion. Since luminance VOP is expanded (and padded) with 16 pixels to each of the four directions, the width and height of the Y plane are 32 pixels larger than those of the VOL, respectively.

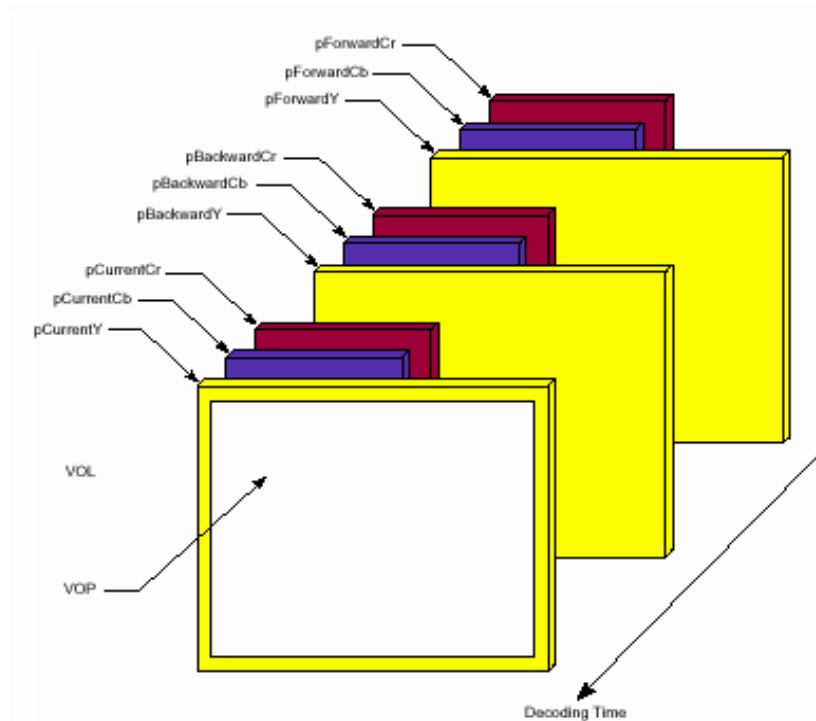
The size (Width, Height) of Cb or Cr plane is half the size of Y plane, because chrominance VOPs are expanded with 8 pixels to each direction.

The A plane has the same size as the Y plane.

[Figure 16-38](#) shows the relationship between the pixel plane, VOL, and VOP.

Three sets of pixel planes, each consisting of an Y plane, Cb plane and Cr plane, should be 32-bit word aligned in user allocation. They are referred to as the current, forward and backward pixel planes. Note that the backward pixel plane set is not required in MPEG-4 Simple Profile, because it does not support B-VOP.

**Figure 16-38 Pixel Plane, VOL, and VOP**



## Macroblock Types

Macroblock types in Intra coded (I-), Predictive coded (P-), and Bidirectional Predictive coded (B-) VOP are defined as follows:

```
typedef enum {
    IPP_VIDEO_INTER           = 0, /* P picture or P-VOP */
    IPP_VIDEO_INTER_Q         = 1, /* P picture or P-VOP */
    IPP_VIDEO_INTER4V         = 2, /* P picture or P-VOP */
}
```

```

    IPP_VIDEO_INTRA          = 3,  /* I and P picture, or I- and P-VOP
*/
    IPP_VIDEO_INTRA_Q        = 4,  /* I and P picture, or I- and P-VOP
*/
    IPP_VIDEO_DIRECT         = 6,  /* B picture or B-VOP (MPEG-4 only)
*/
    IPP_VIDEO_INTERPOLATE    = 7,  /* B picture or B-VOP */
    IPP_VIDEO_BACKWARD       = 8,  /* B picture or B-VOP */
    IPP_VIDEO_FORWARD        = 9   /* B picture or B-VOP */
} IppMacroblockType;

```

## Motion Vector

The structure `IppMotionVector` can be used in both MPEG-4 and H.263 video processing functions and is defined as follows:

```

typedef struct {
    Ipp16s dx;
    Ipp16s dy;
} IppMotionVector;

```

Depending on the macroblock type, there are up to eight valid motion vectors (MVs) in a macroblock (MB). Below is the vector manipulation scheme adopted by Intel IPP MPEG-4 codec, where `pMVForward` and `pMVBackward` denote the two vector buffers allocated for each macroblock:

- Two buffers per macroblock in P- or B-VOP, including `pMVForward[4]` and `pMVBackward[4]`;
- Elements are block based, contiguously stored per each buffer;
- In P-VOP, only `pMVForward[]` is used and valid; `pMVBackward[]` is not used;
- If macroblock type is “IPP\_VIDEO\_INTER” or “IPP\_VIDEO\_INTER\_Q”, and if not transparent, `pMVForward[0] ~ [3]` must be filled with the same decoded MV;
- If macroblock is INTRA coded or skipped, `pMVForward[0] ~ [3]` should be padded with zero MVs;
- In B-VOP, `pMVForward[]` and `pMVBackward[]` may or may not be used, which depends on the macroblock type;
- If B\_VOP, if macroblock type is not “IPP\_VIDEO\_DIRECT”, `pMVForward[1] ~ [3]` and `pMVBackward[1] ~ [3]` are NOT used.

Note that coordinates are related to the absolute coordinate system shown in [\[ISO14496\]](#), Figure 7-19.

## Transparent Status

Transparent status is a three-state value in one byte, or `Ipp8u`. The three possible states are defined as:

```
enum {
    IPP_VIDEO_TRANSPARENT    = 0, /* Wholly transparent */
    IPP_VIDEO_PARTIAL        = 1, /* Partially transparent */
    IPP_VIDEO_OPAQUE         = 2, /* Opaque */
};
```

Transparent status is block based in MPEG-4. Thus,

- One buffer per macroblock;
- Elements are block based, contiguously stored;
- One byte (`Ipp8u`) per element for one block;
- Four elements per macroblock;
- The first element (for block 0) must be 32-bit-aligned (which should be ensured by the user);
- Macroblock transparent status can be determined by evaluating the value of the whole word.

## Quantization Parameter

Quantization parameters (QPs) of intra-coded macroblocks should be stored in order to perform DC and AC prediction for the intra-coded macroblocks spatially to the right and/or below if they exist. The storage buffer for these parameters must be as follows:

- One row buffer per I- and P-VOP;
- The buffer is used for coefficient prediction;
- Before decoding an “intra” or “intra+q” macroblock, the buffer saves the QPs of the MBs of the upper macroblock row;
- After an “intra” or “intra+q” macroblock is decoded, the corresponding QP (of the macroblock spatially above) should be updated by the current QP;
- Each element is one byte (`Ipp8u`) for one macroblock.

## Direction

Direction is relevant when performing DC/AC prediction and zigzag scan. The following enumeration is used to represent the direction:

```
enum {  
    IPP_VIDEO_NONE          = 0,  
    IPP_VIDEO_HORIZONTAL    = 1,  
    IPP_VIDEO_VERTICAL      = 2  
};
```

## Rectangle Plane

The following structure is defined in Intel IPP to represent a rectangle:

```
typedef struct _IppiRect {  
    int      x;  
    int      y;  
    int      width;  
    int      height;  
}
```

## Quantization Parameters

The following enumeration indicates values of quantization parameters

```
enum  
{  
    IPP_DCScalerLinear      = 0,  
    IPP_DCScalerNonLinear   = 1,  
};
```

## Coordinates

The following structure indicates values of coordinates for (x, y) point

```
typedef struct_IppCoordinate  
{  
    int      x;  
    int      y;  
}IppCoordinate;
```

## Block Types

The following enumeration indicates block types (4 *Y*-blocks, *U* block, *V* block, 4 *Alpha* blocks):

```
typedef enum _BlockNum {
    Y_BLOCK1                = 0,
    Y_BLOCK2                = 1,
    Y_BLOCK3                = 2,
    Y_BLOCK4                = 3,
    U_BLOCK                 = 4,
    V_BLOCK                 = 5,
    A_BLOCK1                = 6,
    A_BLOCK2                = 7,
    A_BLOCK3                = 8,
    A_BLOCK4                = 9,
} BlockNum;
```

## Bilinear Interpolation Type

The following enumeration indicates bilinear interpolation type

```
enum {
    IPP_VIDEO_INTEGER_PIXEL    = 0,
    IPP_VIDEO_HALF_PIXEL_X    = 1,
    IPP_VIDEO_HALF_PIXEL_Y    = 1,
    IPP_VIDEO_HALF_PIXEL_XY    = 1,
};
```

## Binary Alpha Block Type

The following enumeration indicates binary alpha block type

```
typedef enum _BAB_TYPE {
    MVDZ_NOUPDT                = 0,
    MVDNZ_NOUPDT                = 1,
    ALL_TRANSP                  = 2,
    ALL_OPAQUE                  = 3,
    INTRA_CAE                   = 4,
    INTRA_CAE_MVDZ              = 5,
```



```
        INTRA_CAE_MVDNZ                = 6,
    } iPPBABType;
```

Block Type

The following enumeration indicates the block type:

```
enum {
    IPPVC_BLOCK_LUMA,
    IPPVC_BLOCK_CHROMA
};
```

Sprite Type

The following enumeration indicates the sprite type:

```
enum {
    IPPVC_SPRITE_STATIC = 1,
    IPPVC_SPRITE_GMC = 2
};
```

Buffers

This section describes buffers and their layout required by Intel IPP. You must allocate and/or initialize the buffers according to the following specifications:

**Video plane buffers.** You should allocate buffers to store the decoded picture (H.263) or video object (MPEG-4) consisting of texture components (Y/Cb/Cr) and alpha components (binary/grayscale) if shape coding is supported. The presence of each component depends on the VOL shape type according to `video_object_layer_shape`. The following table details the dependency, where “x” means required and “-” means not required.

Table 16-18 Video Buffer Allocation Requirements

VOL Shape Type	Texture		
	Y/Cb/Cr Planes	Binary Alpha Plane	Grayscale Alpha Plane
rectangular	x	-	-
binary	x	x	-
binary only	-	x	-
grayscale	x	x	x

Two sets of these buffers should be available: one for the current picture or video object, the other for the previous (forward-reference) picture or video object; an additional set of buffers should also be allocated if bi-directional prediction (B-VOP) is supported. A more detailed description of the video planes could be found in [Pixel Planes and Alpha Plane](#) section in this chapter.

**Motion vector buffers.** A motion vector buffer contains four elements of `IppMotionVector` data. Elements are block based, contiguously stored per each buffer. You should allocate one MV buffer per macroblock for P-VOP; if B-VOP is supported, two MV buffers should be available for bi-directional prediction.

If macroblock type is `IPP_VIDEO_INTER` or `IPP_VIDEO_INTER_Q`, but not `IPP_VIDEO_INTER4V`, and if it is not transparent, the four elements must be filled with the same MV.

If a macroblock is INTRA coded or skipped, the four elements should be padded with zero MVs.

In B-VOP, the forward or backward MV buffer may or may not be used, which depends on the macroblock type.

If macroblock type is not `IPP_VIDEO_DIRECT` in B-VOP, elements [1]~[3] of either buffers are NOT used; only elements [0] may be used.

**Transparent status buffer.** You should allocate one transparent status buffer for the macroblock. Elements, of type `Ipp8u`, are block based and stored contiguously. There are four elements for a macroblock, corresponding to the 16x16 area.

Note that the first element (for block 0) must be 32-bit aligned, which should be ensured by the user. MB transparent status is then detected by evaluating the value of the whole word.

**Quantization parameter buffer.** You should allocate one “row buffer” storing the quantization parameters (QPs) for the P-VOP with one byte (`Ipp8u`) per each element. This buffer is used for coefficient prediction. Before decoding an INTRA coded macroblock, the corresponding element in the QP buffer saves the QP of the macroblock of the upper MB row. After a macroblock is decoded, the corresponding element (the element storing QP of the MB above) should be updated by the QP of the current macroblock.

**Coefficient buffers.** You should allocate two coefficient buffers for Intra DC/AC prediction: a row buffer that contains  $((\text{mb\_num\_per\_row} * 2 + 1) * 8)$  elements of `Ipp16s` type, and a column buffer that contains 16 elements of `Ipp16s` type.

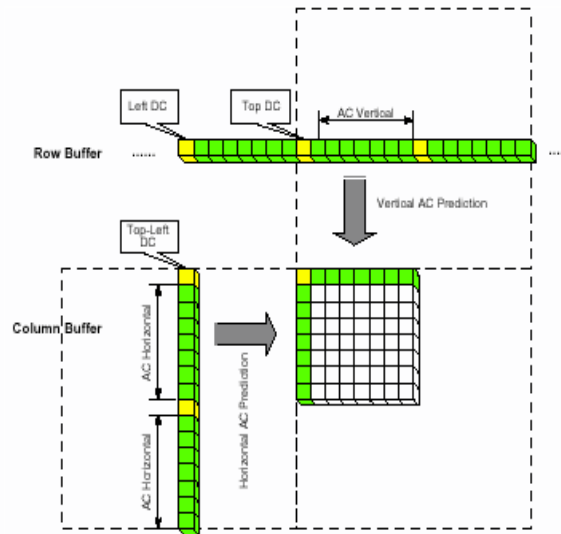
Every 8 elements of both row and column buffers, plus one element 8 units ahead in the row buffer, are used to perform DC/AC prediction for an INTRA coded block in a macroblock. Each group of them stores the coefficient predictors of the neighbor block spatially upper or left to the

block currently to be decoded. Within every 8 elements, the first one stores the DC coefficient and the others store quantized AC coefficients. A negative-valued DC coefficient signals that this neighbor block is not INTRA coded, thus neither the DC nor the AC coefficients are valid.

All DC elements in row buffer should be initialized to -1 prior to decoding each VOP; in addition, the two DC elements in column buffer should also be initialized to -1 prior to decoding each macroblock row.

The detailed coefficient buffer layout is illustrated in [Figure 16-39](#).

**Figure 16-39 Coefficient Buffer Layout Chart**



DC Coefficient Prediction:

$$QF_x[0][0] = PQF_x[0][0] + Fp[0][0] // dc\_scaler$$

AC Coefficient Prediction:

$$QF_x[v][0] = PQF_x[v][0] + (QF_p[v][0] * QPp) // QP_x \quad v = 1 \text{ to } 7$$

or:

$$QF_x[0][u] = PQF_x[0][u] + (QF_p[0][u] * QPp) // QP_x \quad u = 1 \text{ to } 7$$

API:

```
IppStatus ippiPredictReconCoefIntra_MPEG4_16s(Ipp16s* pSrcDst,
Ipp16s* pPredBufRow, Ipp16s* pPredBufCol, int curQP, int predQP,
int predDir, int ACPredFlag, IppVideoComponent videoComp);
```

## Motion Compensation

---

### Copy8x8QP\_MPEG4, Copy16x8QP\_MPEG4, Copy16x16QP\_MPEG4

*Copy fixed size blocks with quarter-pixel accuracy.*

---

#### Syntax

```
IppStatus ippiCopy8x8QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*  
    pDst, int dstStep, int acc, int rounding);  
IppStatus ippiCopy16x8QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*  
    pDst, int dstStep, int acc, int rounding);  
IppStatus ippiCopy16x16QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*  
    pDst, int dstStep, int acc, int rounding);
```

#### Parameters

<i>pSrc</i>	Pointer to the source block.
<i>srcStep</i>	Step in bytes through the source plane.
<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Step in bytes through the destination plane.
<i>acc</i>	Parameter that determines quarter-pixel accuracy.
<i>rounding</i>	Parameter that determines type of rounding for pixel approximation; may be 0 or 1.

#### Description

The functions `ippiCopy8x8QP_MPEG4_8u_C1R`, `ippiCopy16x8QP_MPEG4_8u_C1R`, and `ippiCopy16x16QP_MPEG4_8u_C1R` are declared in the `ippvc.h` file. These functions copy the source block to the destination block with half-pixel accuracy. Parameter `rounding` has the same meaning as `vop_rounding_type` in [ISO14496]. Parameter `acc` is a four-bit value, the bits 0-1 define quarter-pixel offset in horizontal direction and bits 2-3 define quarter-pixel offset in vertical direction. The process of quarter-pixel interpolation is described in [ISO14496] subclause 7.6.2.2.

These functions are used in the MPEG-4 encoder and decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

### Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error when at least one input pointer is NULL.

---

## OBMC8x8HP\_MPEG4, OBMC16x16HP\_MPEG4, OBMC8x8QP\_MPEG4,

*Perform overlapped block motion compensation (OBMC) for one luminance block.*

---

### Syntax

```
IppStatus ippIOBMC8x8HP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, const IppMotionVector* pMVCur, const IppMotionVector*
    pMVLeft, const IppMotionVector* pMVRight, const IppMotionVector* pMVAbove,
    const IppMotionVector* pMVBelow, int rounding);

IppStatus ippIOBMC16x16HP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, const IppMotionVector* pMVCur, const IppMotionVector*
    pMVLeft, const IppMotionVector* pMVRight, const IppMotionVector* pMVAbove,
    const IppMotionVector* pMVBelow, int rounding);

IppStatus ippIOBMC8x8QP_MPEG4_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
    pDst, int dstStep, const IppMotionVector* pMVCur, const IppMotionVector*
    pMVLeft, const IppMotionVector* pMVRight, const IppMotionVector* pMVAbove,
    const IppMotionVector* pMVBelow, int rounding);
```

### Parameters

`pSrc` Pointer to the source blockcoallocated with the destination block.

`srcStep` Width in bytes of the source plane.

`pDst` Pointer to the destination block.

`dstStep` Width in bytes of the destination plane.

`pMVCur` Pointer to the motion vector for the current block.

<i>pMVLeft</i>	Pointer to the motion vector for the left block.
<i>pMVRight</i>	Pointer to the motion vector for the right block.
<i>pMVAbove</i>	Pointer to the motion vector for the above block.
<i>pMVBelow</i>	Pointer to the motion vector for the below block.
<i>rounding</i>	Parameter that determines type of rounding for half pixel approximation; may be 0 or 1.

## Description

The functions `ippiOBMC8x8HP_MPEG4_8u_C1R`, `ippiOBMC16x16HP_MPEG4_8u_C1R`, and `ippiOBMC8x8QP_MPEG4_8u_C1R` are declared in the `ippvc.h` file. The functions `ippiOBMC8x8HP_MPEG4_8u_C1R` and `ippiOBMC16x16HP_MPEG4_8u_C1R` perform the overlapped block motion compensation of the *pSrc* with the half-pixel accuracy ([ISO14496], subclause 7.6.2.1) and `ippiOBMC8x8QP_MPEG4_8u_C1R` - with quarter-pixel accuracy ([ISO14496], subclause 7.6.2.2) and store the results in the destination block *pDst*. Overlapped motion compensation is specified in [ISO14496], subclause 7.6.6.

`ippiOBMC16x16HP_MPEG4_8u_C1R` should be used for overlapped motion compensation in Reduced Resolution VOPs as specified in [ISO14496], subclause 7.6.10.

The function `ippiOBMC8x8HP_MPEG4_8u_C1R` is used in the H.263 and MPEG-4 encoders and decoders included into IPP Samples. The function `ippiOBMC8x8QP_MPEG4_8u_C1R` is used in the MPEG-4 encoder and decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

## Sprite and Global Motion Compensation

Functions for sprite and Global Motion Compensation (GMC) are used in decoding processes of static sprites and in decoding/encoding processes of Video Object Layers with GMC.

---

## WarpInit\_MPEG4

*Initializes IppiWarpSpec\_MPEG4 structure for further usage in GMC or Sprite reconstruction.*

---

### Syntax

```
IppStatus ippiWarpInit_MPEG4(IppiWarpSpec_MPEG4 *pSpec, const int *pDU, const int *pDV, int numWarpingPoints, int spriteType, int warpingAccuracy, int roundingType, int quarterSample, int fcode, const IppiRect* pSpriteRect, const IppiRect* pVopRect);
```

### Parameters

<i>pSpec</i>	Pointer to IppiWarpSpec_MPEG4 structure.
<i>pDU</i>	Pointer to an array of x-coordinate of warping points.
<i>pDV</i>	Pointer to an array of y-coordinate of warping points.
<i>numWarpingPoints</i>	Number of warping points, valid in [0-4].
<i>spriteType</i>	Indicates a sprite coding mode, takes values IPPVC_SPRITE_STATIC, indicating static sprites IPPVC_SPRITE_GMC, indicating GMC.
<i>warpingAccuracy</i>	Accuracy of warping, valid in [0-3].
<i>roundingType</i>	Parameter that determines type of rounding for pixel approximation; may be 0 or 1.
<i>quarterSample</i>	Parameter that indicates a quarter sample mode; may be 0 or 1.
<i>fcode</i>	Parameter that determines the range of motion vector, valid in the range [1,7].
<i>pSpriteRect</i>	Parameter that determines rectangle region for Sprite (or reference VOP for GMC).
<i>pVopRect</i>	Parameter that determines rectangle region for current VOP.

## Description

The function `ippiWarpInit_MPEG4` is declared in the `ippvc.h` file. This function initializes `IppiWarpSpec_MPEG4` structure for further usage in GMC or Sprite reconstruction. The meaning of parameters is described in [\[ISO14496\]](#) subclause 7.8.

This function is used in the MPEG-4 encoder and decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsSizeErr</code>	Indicates no error when width or height of images is less than or equal to zero.
<code>ippStsOutOfRangeErr</code>	Indicates an error when <i>numWarpingPoints</i> , <i>warpingAccuracy</i> , or <i>fcode</i> are out of the valid range.

---

## WarpGetSize\_MPEG4

*Returns size of `IppiWarpSpec_MPEG4` structure.*

---

## Syntax

```
IppStatus ippiWarpGetSize_MPEG4(int* pSpecSize);
```

## Parameters

*pSpecSize*                      Pointer to the resulting size of the structure `IppiWarpSpec_MPEG4`.

## Description

The function `ippiWarpGetSize_MPEG4` is declared in the `ippvc.h` file. This function returns a size of structure `IppiWarpSpec_MPEG4`.

This function is used in the MPEG-4 encoder and decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when pointer <i>pSpecSize</i> is NULL.

## WarpLuma\_MPEG4

*Warps arbitrary rectangular luminance region.*

### Syntax

```
IppStatus ippWarpLuma_MPEG4_8u_C1R(const Ipp8u* pSrcY, int srcStepY, Ipp8u*
    pDstY, int dstStepY, const IppiRect* pDstRect, const IppiWarpSpec_MPEG4*
    pSpec);
```

### Parameters

<i>pSrcY</i>	Pointer to the origin of the source plane.
<i>srcStep</i>	Step in bytes through the source plane.
<i>pDst</i>	Pointer to the destination region.
<i>dstStep</i>	Step in bytes through the destination plane.
<i>pDstRect</i>	Rectangular destination region.
<i>pSpec</i>	Pointer to the structure with motion parameters.

### Description

The function `ippWarpLuma_MPEG4_8u_C1R` is declared in the `ippvc.h` file. This function warps an arbitrary rectangular area using motion information from `IppiWarpSpec_MPEG4` structure. This function may be used in GMC or Sprite reconstruction ([ISO14496] subclause 7.8).

This function is used in the MPEG-4 encoder and decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## WarpChroma\_MPEG4

*Warps arbitrary rectangular chrominance region.*

---

### Syntax

```
IppStatus ippWarpChroma_MPEG4_8u_P2R(const Ipp8u *pSrcCb, int srcStepCb, const  
    Ipp8u *pSrcCr, int srcStepCr, Ipp8u* pDstCb, int dstStepCb, Ipp8u* pDstCr,  
    int dstStepCr, const IppiRect* pDstRect, const IppiWarpSpec_MPEG4 *pSpec);
```

### Parameters

<i>pSrcCb</i>	Pointer to the origin of the first source plane.
<i>srcStepCb</i>	Step in bytes through the first source plane.
<i>pSrcCr</i>	Pointer to the origin of the second source plane.
<i>srcStepCr</i>	Step in bytes through the second source plane.
<i>pDstCb</i>	Pointer to the first destination plane.
<i>dstStepCb</i>	Step in bytes through the first destination plane.
<i>pDstCr</i>	Pointer to the second destination plane.
<i>dstStepCr</i>	Step in bytes through the second destination plane.
<i>pDstRect</i>	Rectangular destination region.
<i>pSpec</i>	Pointer to the structure with motion parameters.

### Description

The function `ippWarpChroma_MPEG4_8u_P2R` is declared in the `ippvc.h` file. This function warps an arbitrary rectangular chrominance area using motion information from `IppiWarpSpec_MPEG4` structure. This function may be used in GMC or Sprite reconstruction ([ISO14496] subclause 7.8).

This function is used in the MPEG-4 encoder and decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .

---

## CalcGlobalMV\_MPEG4

*Calculates Global Motion Vector for one macroblock.*

---

### Syntax

```
IppStatus ippCalcGlobalMV_MPEG4(int xOffset, int yOffset, IppMotionVector*
    pGMV, const IppiWarpSpec_MPEG4* pSpec);
```

### Parameters

<i>xOffset</i>	The left coordinate of top-left corner of luma 16x16 block.
<i>yOffset</i>	The top coordinate of top-left corner of luma 16x16 block.
<i>pGMV</i>	Pointer to the resulting motion vector.
<i>pSpec</i>	Pointer to the structure with motion parameters.

### Description

The function `ippCalcGlobalMV_MPEG4` is declared in the `ippvc.h` file. This function calculates a global motion vector for a macroblock and clips it to lie in range  $[-2^{f_{code}+4}, 2^{f_{code}+4}-1]$  as it described in [\[ISO14496\]](#) subclause 7.8.7.3.

This function is used in the MPEG-4 encoder and decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

---

## ChangeSpriteBrightness\_MPEG4

*Change brightness after sprite warping.*

---

### Syntax

```
IppStatus ippChangeSpriteBrightness_MPEG4_8u_C1IR(const Ipp8u *pSrcDst, int
    srcDstStep, int width, int height, int brightnessChangeFactor);
```

### Parameters

<i>pSrcDst</i>	Pointer to the video plane.
<i>srcDstStep</i>	Step in bytes through the video plane.
<i>width</i>	The width of the video plane.
<i>height</i>	The height of the video plane.
<i>brightnessChangeFactor</i>	Factor for changing brightness; valid in the range [-112; 1648].

### Description

The function `ippiChangeSpriteBrightness_MPEG4_8u_C1IR` is declared in the `ippvc.h` file. This function changes brightness of video plane after sprite warping ([[ISO14496](#)] subclause 7.8).

This function is used in the MPEG-4 decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error when <i>brightnessChangeFactor</i> is out of the valid range.

## Motion Vector Decoding and Padding

### DecodePadMV\_PVOP\_MPEG4

*Decodes and pads four motion vectors of the non-intra macroblock in P-VOP.*

#### Syntax

```
IppStatus ippiDecodePadMV_PVOP_MPEG4(Ipp8u** ppBitStream, int* pBitOffset,
    IppMotionVector* pSrcMVLeftMB, IppMotionVector* pSrcMVUpperMB,
    IppMotionVector* pSrcMVUpperRightMB, IppMotionVector* pDstMVCurMB, Ipp8u*
    pTranspLeftMB, Ipp8u* pTranspUpperMB, Ipp8u* pTranspUpperRightMB, Ipp8u*
    pTranspCurMB, int fCodeForward, IppMacroblockType MBType);
```

#### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> will be updated after motion vector decoding.
<i>pBitOffset</i>	Pointer to the bit position of the byte pointed by <i>ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> will be updated after motion vectors decoding.
<i>pSrcMVLeftMB</i>	Pointer to the motion vector buffer of the macroblock spatially at the left side of the current macroblock.
<i>pSrcMVUpperMB</i>	Pointer to the motion vector buffer of the macroblock spatially at the upper side of the current macroblock.
<i>pSrcMVUpperRightMB</i>	Pointer to the motion vector buffer of the macroblock spatially at the upper-right side of the current macroblock.
<i>pDstMVCurMB</i>	Pointer to the destination motion vector buffer of the current macroblock which contains four decoded motion vectors.
<i>pTranspLeftMB</i>	Pointer to the transparent status buffers of the macroblock spatially at the left side of the current macroblock.
<i>pTranspUpperMB</i>	Pointer to the transparent status buffers of the macroblock spatially at the upper side of the current macroblock.
<i>pTranspUpperRightMB</i>	Pointer to the transparent status buffers of the macroblock spatially at the upper-right side of the current macroblock.

<i>pTranspCurMB</i>	Pointer to the transparent status buffers of the current macroblock.
<i>fCodeForward</i>	A code equal to <code>vop_fcode_forward</code> in MPEG-4 bitstream syntax.
<i>MBType</i>	The type of the current macroblock.

## Description

The function `ippiDecodePadMV_PVOP_MPEG4` is declared in the `ippalign.h` file. This function decodes and pads four motion vectors of the non-intra macroblock in P-VOP.

The motion vector decoding process is specified in [\[ISO14496\]](#), subclause 7.6.3. The motion vector prediction is specified in [\[ISO14496\]](#), subclause 7.6.5. The motion vector padding process is specified in [\[ISO14496\]](#), subclause 7.6.1.6.

*pTranspLeftMB*, *pTranspUpperMB*, *pTranspUpperRightMB* should be set to `IPP_VIDEO_TRANSPARENT` if the corresponding macroblock is outside the VOP.

If *MBType* is not equal to `IPP_VIDEO_INTER4V`, the destination motion vector buffer is still filled with the same decoded vector.




---

**NOTE.** The pointer to the transparent status buffer should be 32-bit aligned.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the pointers <i>ppBitStream</i> , <i>*ppBitStream</i> , <i>pBitOffset</i> , <i>pTranspLeftMB</i> , <i>pTranspUpperMB</i> , <i>pTranspUpperRightMB</i> , <i>pTranspCurMB</i> , <i>pDstMVCurMB</i> is NULL.
<code>ippStsMP4BitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsMP4FcodeErr</code>	Indicates an error condition if <i>fCodeForward</i> is out of the range [1, 7].
<code>ippStsMP4AlignErr</code>	Indicates an error condition if at least one of the pointers <i>pTranspLeftMB</i> , <i>pTranspUpperMB</i> , <i>pTranspUpperRightMB</i> , <i>pTranspCurMB</i> is not 32-bit aligned.
<code>ippStsMP4MVCodeErr</code>	Indicates an error condition if illegal Huffman code is detected through the MV stream processing.

---

## DecodeMV\_BVOP\_Forward\_MPEG4

*Decodes motion vector of the macroblock  
in B-VOP forward mode.*

---

### Syntax

```
IppStatus ippDecodeMV_BVOP_Forward_MPEG4(Ipp8u** ppBitStream, int* pBitOffset,
    IppMotionVector* pSrcDstMVF, int fCodeForward);
```

### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> will be updated after motion vector decoding.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> will be updated after motion vectors decoding.
<i>pSrcDstMVF</i>	Pointer to the forward motion vector predictor on input. Pointer to the forward motion vector of the current macroblock on output.
<i>fCodeForward</i>	A code equal to <code>vop_fcode_forward</code> in MPEG-4 bitstream syntax.

### Description

The function `ippDecodeMV_BVOP_Forward_MPEG4` is declared in the `ippalign.h` file. This function decodes motion vector of the macroblock in B-VOP forward mode. The motion vector decoding process is specified in [ISO14496], subclause 7.6.8.

After decoding a macroblock of forward mode, only the forward predictor is set to the decoded forward vector. The forward motion vector predictor should be reset to zero at the beginning of each macroblock row.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the pointers <i>ppBitStream</i> , <i>*ppBitStream</i> , <i>pBitOffset</i> , <i>pSrcDstMVF</i> is NULL.
<code>ippStsMP4BitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsMP4FcodeErr</code>	Indicates an error condition if <i>fCodeForward</i> is not in the range [1,7].

`ippStsMP4MVCodeErr` Indicates an error condition if illegal Huffman code is detected through the MV stream processing.

---

## DecodeMV\_BVOP\_Backward\_MPEG4

*Decodes motion vector of the macroblock in B-VOP backward mode.*

---

### Syntax

```
IppStatus ippDecodeMV_BVOP_Backward_MPEG4(Ipp8u** ppBitStream, int*
    pBitOffset, IppMotionVector* pSrcDstMVB, int fCodeBackward);
```

### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> will be updated after motion vector decoding.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> will be updated after motion vectors decoding.
<i>pSrcDstMVB</i>	Pointer to the backward motion vector predictor on input. Pointer to the backward motion vector of the current macroblock on output.
<i>fCodeBackward</i>	A code equal to <code>vop_fcode_backward</code> in MPEG-4 bitstream syntax.

### Description

The function `ippDecodeMV_BVOP_Backward_MPEG4` is declared in the `ippalign.h` file. This function decodes motion vector of the macroblock in B-VOP backward mode. The motion vector decoding process is specified in [\[ISO14496\]](#), subclause 7.6.8.

After decoding a macroblock of backward mode, only the backward predictor is set to the decoded backward vector. The backward motion vector predictor should be reset to zero at the beginning of each macroblock row.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the pointers <i>ppBitStream</i> , <i>*ppBitStream</i> , <i>pBitOffset</i> , <i>pSrcDstMVB</i> is NULL.



`ippStsMP4BitOffsetErr` Indicates an error condition if `*pBitOffset` is out of the range [0, 7].

`ippStsMP4FcodeErr` Indicates an error condition if `fCodeBackward` is out of the range [1,7].

`ippStsMP4MVCodeErr` Indicates an error condition if illegal Huffman code is detected through the MV stream processing.

---

## DecodeMV\_BVOP\_Interpolate\_MPEG4

*Decodes motion vector of the macroblock  
in B-VOP interpolate mode.*

---

### Syntax

```
IppStatus ippIDecodeMV_BVOP_Interpolate_MPEG4 (Ipp8u** ppBitStream, int*
    pBitOffset, IppMotionVector* pSrcDstMVF, IppMotionVector* pSrcDstMVB, int
    fCodeForward, int fCodeBackward);
```

### Parameters

<code>ppBitStream</code>	Pointer to pointer to the current byte in the bitstream buffer. <code>*ppBitStream</code> will be updated after motion vector decoding.
<code>pBitOffset</code>	Pointer to the bit position in the byte pointed by <code>ppBitStream</code> . Valid within the range 0 to 7. <code>*pBitOffset</code> will be updated after motion vectors decoding.
<code>pSrcDstMVF</code>	Pointer to the forward motion vector predictor on input. Pointer to the forward motion vector of the current macroblock on output.
<code>pSrcDstMVB</code>	Pointer to the backward motion vector predictor on input. Pointer to the backward motion vector of the current macroblock on output.
<code>fCodeForward</code>	A code equal to <code>vop_fcode_forward</code> in MPEG-4 bitstream syntax.
<code>fCodeBackward</code>	A code equal to <code>vop_fcode_backward</code> in MPEG-4 bitstream syntax.

### Description

The function `ippIDecodeMV_BVOP_Interpolate_MPEG4` is declared in the `ippalign.h` file. This function decodes motion vector of the macroblock in B-VOP interpolate mode. The motion vectors decoding process is specified in [\[ISO14496\]](#), subclause 7.6.8.

Both the forward and backward motion vector predictor should be reset to zero at the beginning of each macroblock row.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the pointers <i>ppBitStream</i> , <i>*ppBitStream</i> , <i>pBitOffset</i> , <i>pSrcDstMVF</i> , <i>pSrcDstMVB</i> is NULL.
<code>ippStsMP4BitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsMP4FcodeErr</code>	Indicates an error condition if <i>fCodeForward</i> or <i>fCodeBackward</i> is out of the range [1,7].
<code>ippStsMP4MVCodeErr</code>	Indicates an error condition if illegal Huffman code is detected through the MV stream processing.

---

## DecodeMV\_BVOP\_Direct\_MPEG4

*Decodes motion vectors of the macroblock in B-VOP using direct mode.*

---

### Syntax

```
IppStatus ippIDecodeMV_BVOP_Direct_MPEG4(Ipp8u** ppBitStream, int* pBitOffset,
const IppMotionVector* pSrcMV, IppMotionVector* pDstMVF, IppMotionVector*
pDstMVB, Ipp8u* pTranspSrcMB, int TRB, int TRD);
```

### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> will be updated after motion vector decoding.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> will be updated after motion vectors decoding.
<i>pSrcMV</i>	Pointer to the motion vector buffer of the co-located macroblock in the most recently decoded I- or P-VOP.
<i>pDstMVF</i>	Pointer to the forward motion vector buffer of the current macroblock.
<i>pDstMVB</i>	Pointer to the backward motion vector buffer of the current macroblock.

<i>pTranspSrcMB</i>	Pointer to the transparent status buffer of the co-located macroblock.
<i>TRB</i>	The difference in temporal reference of the B-VOP and the previous reference VOP.
<i>TRD</i>	The difference in temporal reference of the temporally next reference VOP with temporally previous reference VOP.

## Description

The function `ippiDecodeMV_BVOP_Direct_MPEG4` is declared in the `ippalign.h` file. This function decodes motion vectors of the macroblock in B-VOP using direct mode. The source motion vector decoding process is specified in [ISO14496], subclause 7.6.8. The destination motion vectors calculation process is specified in [ISO14496], subclause 7.6.9.




---

**NOTE.** The pointer to the transparent status buffer should be 32-bit aligned.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the pointers <i>ppBitStream</i> , <i>*ppBitStream</i> , <i>pBitOffset</i> , <i>pSrcMV</i> , <i>pDstMVF</i> , <i>pDstMVB</i> , <i>pTranspSrcMB</i> is NULL.
<code>ippStsMP4BitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsMP4TempDiffErr</code>	Indicates an error condition if <i>TRB</i> or <i>TRD</i> has a zero or negative value.
<code>ippStsMP4MVCodeErr</code>	Indicates an error condition if illegal Huffman code is detected through the MV stream processing.

## DecodeMV\_BVOP\_DirectSkip\_MPEG4

*Decodes motion vectors of the macroblock in B-VOP using direct mode when the current macroblock is skipped.*

---

### Syntax

```
IppStatus ippDecodeMV_BVOP_DirectSkip_MPEG4 (const IppMotionVector* pSrcMV,
        IppMotionVector* pDstMVF, IppMotionVector* pDstMVB, Ipp8u* pTranspSrcMB, int
        TRB, int TRD);
```

### Parameters

<i>pSrcMV</i>	Pointer to the motion vector buffer of the co-located macroblock in the most recently decoded I- or P-VOP.
<i>pTranspSrcMB</i>	Pointer to the transparent status buffer of the co-located macroblock.
<i>pDstMVF</i>	Pointer to the forward motion vector buffer of the current macroblock.
<i>pDstMVB</i>	Pointer to the backward motion vector buffer of the current macroblock.
<i>TRB</i>	The difference in temporal reference of the B-VOP and the previous reference VOP.
<i>TRD</i>	The difference in temporal reference of the temporally next reference VOP with temporally previous reference VOP, assuming B-VOPs or skipped VOPs in between.

### Description

The function `ippDecodeMV_BVOP_DirectSkip_MPEG4` is declared in the `ippalign.h` file. This function decodes motion vectors of the macroblock in B-VOP using direct mode when the current macroblock is skipped (that is, `modb == "1"`). The destination motion vectors calculation process is specified in [\[ISO14496\]](#), subclause 7.6.9.5.2. Since macroblock is skipped, *MVD<sub>x</sub>* and *MVD<sub>y</sub>* are set to zero.




---

**NOTE.** The pointer to the transparent status buffer should be 32-bit aligned.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the pointers <i>pSrcMV</i> , <i>pDstMV</i> , <i>pDstMVB</i> , <i>pTranspSrcMB</i> is NULL.
<code>ippStsMP4TempDiffErr</code>	Indicates an error condition if <i>TRB</i> or <i>TRD</i> has a zero or negative value.

## LimitMVToRect\_MPEG4

*Limits the motion vector of current block/macroblock into the extended bounding rectangle.*

### Syntax

```
IppStatus ippLimitMVToRect_MPEG4(const IppMotionVector* pSrcMV,
    IppMotionVector* pDstMV, IppiRect* pRectVOPRef, int xcoord, int ycoord, int
    size);
```

### Parameters

<i>pSrcMV</i>	Pointer to the motion vector of current block or macroblock.
<i>pRectVOPRef</i>	Pointer to the bounding rectangle.
<i>pDstMV</i>	Pointer to the limited motion vector.
<i>xcoord</i> , <i>ycoord</i>	The top-left coordinates of the current block or macroblock.
<i>size</i>	The size of block (8) or macroblock (16).

### Description

The function `ippLimitMVToRect_MPEG4` is declared in the `ippalign.h` file. This function limits the motion vector of current block/macroblock into the extended bounding rectangle as specified in [ISO14496], subclause 7.6.4.

The value of *size* is either 8 for a block or 16 for a macroblock. The width and the height of the bounding rectangle should be larger than twice the value of *size*.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsMP4BlockSizeErr</code>	Indicates an error condition if <i>size</i> is not equal to 8 for block and to 16 for macroblock.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>pRectVOPRef</i> size is incorrect: width or height of the rectangle has a zero or negative value, or is less than $2 * size$ .

## Coefficient Prediction and Reconstruction

---

### PredictReconCoefIntra\_MPEG4

*Performs adaptive DC/AC coefficient prediction for an intra block.*

---

#### Syntax

```
IppStatus ippIPredictReconCoefIntra_MPEG4_16s(Ipp16s* pSrcDst, Ipp16s*
    pPredBufRow, Ipp16s* pPredBufCol, int curQP, int predQP, int predDir, int
    ACPredFlag, IppVideoComponent videoComp);
```

#### Parameters

<i>pSrcDst</i>	Pointer to the coefficient buffer which contains the quantized coefficient residuals (PQFs) of the current block on input and the quantized coefficients (QFs) of the current block on output.
<i>pPredBufRow</i>	Pointer to the first element of the spatially upper block in the corresponding coefficient row buffer (this row buffer will be updated after prediction). See <a href="#">Buffers</a> in this chapter for details.
<i>pPredBufCol</i>	Pointer to the first element of the spatially left block in the corresponding coefficient column buffer (this column buffer will be updated after prediction). See <a href="#">Buffers</a> in this chapter for details.
<i>curQP</i>	Quantization parameter of the current block.
<i>predQP</i>	Quantization parameter of the predictor block.
<i>predDir</i>	Indicates the prediction direction which takes one of the following values:  <code>IPP_VIDEO_HORIZONTAL</code> Predicts horizontally.

---

	<code>IPP_VIDEO_VERTICAL</code>	Predicts vertically.
<i>ACPredFlag</i>	A flag indicating if AC prediction should be performed. It is equal to <code>ac_pred_flag</code> in the bitstream syntax of MPEG-4.	
<i>videoComp</i>	Video component type (luminance, chrominance or alpha) of the current block.	

## Description

The function `ippiPredictReconCoefIntra_MPEG4_16s` is declared in the `ippalign.h` file. This function performs adaptive DC/AC coefficient prediction for an intra block. The DC coefficient prediction is specified in [\[ISO14496\]](#) subclause 7.4.3.2. The AC coefficient prediction is specified in [\[ISO14496\]](#), subclause 7.4.3.3.

Prior to the function call, prediction direction *predDir* should be selected as specified in [\[ISO14496\]](#), subclause 7.4.3.1.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the pointers <i>pSrcDst</i> , <i>pPredBufRow</i> , <i>pPredBufCol</i> is NULL.
<code>ippStsMP4QPErr</code>	Indicates an error condition if <i>curQP</i> or <i>predQO</i> is out of the range [1, 31].
<code>ippStsMP4PredDirErr</code>	Indicates an error condition if <i>predDir</i> is not valid.

## Motion Padding

---

### PadMBHorizontal\_MPEG4

*Performs horizontal extended padding process on exterior macroblock immediately next to boundary macroblocks.*

---

#### Syntax

```
IppStatus ippPadMBHorizontal_MPEG4_8u(const Ipp8u* pSrcY, const Ipp8u* pSrcCb,
    const Ipp8u* pSrcCr, const Ipp8u* pSrcA, Ipp8u* pDstY, Ipp8u* pDstCb, Ipp8u*
    pDstCr, Ipp8u* pDstA, int stepYA, int stepCbCr);
```

#### Parameters

<i>pSrcY</i>	Pointer to one of the vertical border of the boundary luminance blocks that are chosen to pad the exterior macroblock.
<i>pSrcCb</i>	Pointer to one of the vertical border of the boundary Cb block that is chosen to pad the exterior macroblock.
<i>pSrcCr</i>	Pointer to one of the vertical border of the boundary Cr block that is chosen to pad the exterior macroblock.
<i>pSrcA</i>	Pointer to one of the vertical border of the boundary alpha blocks that are chosen to pad the exterior macroblock.
<i>pDstY</i>	Pointer to the padded exterior luminance blocks.
<i>pDstCb</i>	Pointer to the padded exterior Cb block.
<i>pDstCr</i>	Pointer to the padded exterior Cr block.
<i>pDstA</i>	Pointer to the padded exterior alpha blocks.
<i>stepYA</i>	Width in bytes of the luminance and/or alpha planes.
<i>stepCbCr</i>	Width in bytes of the chrominance planes.



## Description

The function `ippiPadMBHorizontal_MPEG4_8u` is declared in the `ippalign.h` file. This function performs horizontal extended padding process on exterior macroblock immediately next to boundary macroblocks as it is specified in [ISO14496], subclause 7.6.1.3.

If `pSrcA` has a zero value, no alpha plane is available. Otherwise, alpha plane should be padded.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the pointers <code>pSrcY</code> , <code>pSrcCb</code> , <code>pSrcCr</code> , <code>pDstY</code> , <code>pDstCb</code> , <code>pDstCr</code> is NULL, or if <code>pDstA</code> is NULL but <code>pSrcA</code> is not NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <code>stepYA</code> is less than 16 or <code>stepCbCr</code> is less than 8, or at least one of them is not divisible by 4.

---

## PadMBVertical\_MPEG4

*Performs vertical extended padding process on exterior macroblock immediately next to boundary macroblocks.*

---

## Syntax

```
IppStatus ippiPadMBVertical_MPEG4_8u(const Ipp8u* pSrcY, const Ipp8u* pSrcCb,
    const Ipp8u* pSrcCr, const Ipp8u* pSrcA, Ipp8u* pDstY, Ipp8u* pDstCb, Ipp8u*
    pDstCr, Ipp8u* pDstA, int stepYA, int stepCbCr);
```

## Parameters

<code>pSrcY</code>	Pointer to one of the horizontal border of the boundary luminance blocks that are chosen to pad the exterior macroblock.
<code>pSrcCb</code>	Pointer to one of the horizontal border of the boundary Cb block that is chosen to pad the exterior macroblock.
<code>pSrcCr</code>	Pointer to one of the horizontal border of the boundary Cr block that is chosen to pad the exterior macroblock.

<i>pSrcA</i>	Pointer to one of the horizontal border of the boundary alpha blocks that are chosen to pad the exterior macroblock.
<i>pDstY</i>	Pointer to the padded exterior luminance blocks.
<i>pDstCb</i>	Pointer to the padded exterior Cb block.
<i>pDstCr</i>	Pointer to the padded exterior Cr block.
<i>pDstA</i>	Pointer to the padded exterior alpha blocks.
<i>stepYA</i>	Width in bytes of the luminance and/or alpha planes.
<i>stepCbCr</i>	Width in bytes of the chrominance planes.

### Description

The function `ippiPadMBVertical_MPEG4` is declared in the `ippalign.h` file. This function performs vertical extended padding process on exterior macroblock immediately next to boundary macroblocks as it is specified in [\[ISO14496\]](#), subclause 7.6.1.3.

If *pSrcA* has a zero value, no alpha plane is available. Otherwise, alpha plane should be padded.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the pointers <i>pSrcY</i> , <i>pSrcCb</i> , <i>pSrcCr</i> , <i>pDstY</i> , <i>pDstCb</i> , <i>pDstCr</i> is NULL, or if <i>pDstA</i> is NULL but <i>pSrcA</i> is not NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>stepYA</i> is less than 16 or <i>stepCbCr</i> is less than 8, or at least one of them is not divisible by 4.

---

## PadMBGray\_MPEG4

*Fills gray value in exterior macroblock that is not located next to any boundary macroblocks.*

---

### Syntax

```
IppStatus ippiPadMBGray_MPEG4_8u(Ipp8u grayVal, Ipp8u* pDstY, Ipp8u* pDstCb,
    Ipp8u* pDstCr, Ipp8u* pDstA, int stepYA, int stepCbCr);
```

## Parameters

<code>grayVal</code>	The gray value to fill the exterior macroblock/block.
<code>pDstY</code>	Pointer to the padded exterior luminance blocks.
<code>pDstCb</code>	Pointer to the padded exterior Cb block.
<code>pDstCr</code>	Pointer to the padded exterior Cr block.
<code>pDstA</code>	Pointer to the padded exterior alpha blocks.
<code>stepYA</code>	Width in bytes of the luminance and/or alpha planes.
<code>stepCbCr</code>	Width in bytes of the chrominance planes.

## Description

The function `ippiPadMBGray_MPEG4_8u` is declared in the `ippalign.h` file. This function fills gray value in exterior macroblock that is not located next to any boundary macroblocks.

If `pDstA` has a zero value, no alpha plane is available. Otherwise, alpha plane should be padded.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the pointers <code>pDstY</code> , <code>pDstCb</code> , <code>pDstCr</code> is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <code>stepYA</code> is less than 16 or <code>stepCbCr</code> is less than 8, or at least one of them is not divisible by 4.

---

## PadCurrent\_16x16\_MPEG4, PadCurrent\_8x8\_MPEG4

*Perform horizontal and vertical repetitive padding process on luminance/alpha macroblock or chrominance block.*

---

## Syntax

```
IppStatus ippiPadCurrent_16x16_MPEG4_8u_I(Ipp8u* pSrcDst, int step, const  
Ipp8u* pBAB);
```

```
IppStatus ippiPadCurrent_8x8_MPEG4_8u_I(Ipp8u* pSrcDst, int step, const Ipp8u*
pBAB);
```

## Parameters

<i>pSrcDst</i>	Pointer to the macroblock/block to be padded.
<i>step</i>	Width in bytes of the source plane.
<i>pBAB</i>	Pointer to the binary alpha block buffer (BAB). The buffer contains 256 bytes (one byte for one pixel's property).

## Description

The functions `ippiPadCurrent_16x16_MPEG4_8u_I` and `ippiPadCurrent_8x8_MPEG4_8u_I` are declared in the `ippalign.h` file. These functions perform horizontal and vertical repetitive padding process on luminance/alpha 16x16 macroblock and on chrominance 8x8 block, respectively. The horizontal and vertical repetitive padding processes are specified in [\[ISO14496\]](#), subclauses 7.6.1.1 and 7.6.1.2. For chrominance components, the BAB should be generated by subsampling the shape block of the corresponding luminance component.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>step</i> is less than 16 (16x16 version) or 8 (8x8 version) or is not divisible by 4.
<code>ippStsMP4ZeroBABErr</code>	Indicates an error condition if all BAB elements have a zero value.

---

## Vector Padding

---

### PadMV\_MPEG4

*Performs vector padding for a non-transparent macroblock.*

---

#### Syntax

```
IppStatus ippPadMV_MPEG4 (IppMotionVector* pSrcDstMV, Ipp8u* pTransp);
```

#### Parameters

<i>pSrcDstMV</i>	Pointer to the motion vector buffer of the current macroblock to be padded.
<i>pTransp</i>	Pointer to the transparent status buffer of the current macroblock.

#### Description

The function `ippPadMV_MPEG4` is declared in the `ippalign.h` file. This function performs vector padding for a non-transparent macroblock. Motion vector padding process is specified in [\[ISO14496\]](#), subclass 7.6.1.6.



---

**NOTE.** The pointer to the transparent status buffer should be 32-bit aligned.

---

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

## Inverse Quantization

---

### QuantInvIntraInit\_MPEG4, QuantInvInterInit\_MPEG4

*Initialize specification structures.*

---

#### Syntax

```
IppStatus ippiQuantInvIntraInit_MPEG4(const Ipp8u* pQuantMatrix,  
    IppiQuantInvIntraSpec_MPEG4* pSpec, int bitsPerPixel);  
  
IppStatus ippiQuantInvInterInit_MPEG4(const Ipp8u* pQuantMatrix,  
    IppiQuantInvInterSpec_MPEG4* pSpec, int bitsPerPixel);
```

#### Parameters

<i>pQuantMatrix</i>	Pointer to quantization matrix size of 64.
<i>pSpec</i>	Pointer to the initialized specification structure IppiQuantInvIntraSpec_MPEG4 or IppiQuantInvInterSpec_MPEG4.
<i>bitsPerPixel</i>	Video data precision used for saturation of the result. Valid within the range [4, 12].

#### Description

The functions `ippiQuantInvIntraInit_MPEG4` and `ippiQuantInvInterInit_MPEG4` are declared in the `ippvc.h` file. These functions initialize specification structure `IppiQuantInvIntraSpec_MPEG4` or `IppiQuantInvInterSpec_MPEG4`. If *pQuantMatrix* is NULL, the second quantization method is used; otherwise, the first method is used.

These functions are used in the MPEG-4 encoder and decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error pointer <i>pSpec</i> is NULL.

---

<code>ippStsOutOfRangeErr</code>	Indicates an error condition when <i>bitsPerPixel</i> is out of the range [4, 12].
----------------------------------	--

---

## QuantInvIntraGetSize\_MPEG4, QuantInvInterGetSize\_MPEG4

*Return size of specification structures.*

---

### Syntax

```
IppStatus ippiQuantInvIntraGetSize_MPEG4(int* pSpecSize);  
IppStatus ippiQuantInvInterGetSize_MPEG4(int* pSpecSize);
```

### Parameters

<i>pSpecSize</i>	Pointer to the resulting size of the initialized specification structure <code>IppiQuantInvIntraSpec_MPEG4</code> or <code>IppiQuantInvInterSpec_MPEG4</code> .
------------------	---

### Description

The functions `ippiQuantInvIntraGetSize_MPEG4` and `ippiQuantInvInterGetSize_MPEG4` are declared in the `ippvc.h` file. These functions return a size of specification structure `IppiQuantInvIntraSpec_MPEG4` or `IppiQuantInvInterSpec_MPEG4`.

These functions are used in the MPEG-4 encoder and decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error pointer <i>pSpecSize</i> is NULL.

---

## QuantInvIntra\_MPEG4, QuantInvInter\_MPEG4

*Perform inverse quantization on intra/inter coded block.*

---

### Syntax

```
IppStatus ippiQuantInvIntra_MPEG4_16s_C1I(Ipp16s* pCoeffs, int indxLastNonZero,  
    const IppiQuantIntraSpec_MPEG4* pSpec, int QP, int blockType);  
  
IppStatus ippiQuantInvInter_MPEG4_16s_C1I(Ipp16s* pCoeffs, int indxLastNonZero,  
    const IppiQuantInterSpec_MPEG4* pSpec, int QP);  
  
IppStatus ippiQuantInvIntra_MPEG4_16s_I (ipp16s* pSrcDst, int QP, const Ipp8u*  
    pQPMatrix, IppVideoComponent videoComp);  
  
IppStatus ippiQuantInvInter_MPEG4_16s_I (Ipp16s* pSrcDst, int QP, const Ipp8u*  
    pQPMatrix);
```

### Parameters

<i>pCoeffs</i>	Pointer to the quantized DCT coefficients.
<i>indxLastNonZero</i>	Index of the last non-zero coefficient. This parameter provides faster operation. If the value is unknown, set to 63.
<i>pSpec</i>	Pointer to the initialized specification structure IppiQuantInvIntraSpec_MPEG4 or IppiQuantInvInterSpec_MPEG4 initialized by ippiQuantInvIntraInit_MPEG4 or ippiQuantInvInterInit_MPEG4 respectively.
<i>QP</i>	Quantization parameter.
<i>blockType</i>	Indicates the block type. Takes one of the following values: IPPVC_BLOCK_LUMA - for luma and alpha blocks, IPPVC_BLOCK_CHROMA - for chroma blocks.
<i>pSrcDst</i>	Pointer to the coefficient buffer of intra/inter block. Contains quantized coefficients (QF) on input and dequantized coefficients (F) on output.
<i>pQPMatrix</i>	Pointer to the quantization weighting matrix. If <i>pQPMatrix</i> is NULL, this function will use the second inverse quantization method; otherwise, it will use the first method.



`videoComp` (Intra version only) Video component type of the current block. It is used for DC coefficient dequantizing. Takes one of the following flags:

```
IPP_VIDEO_LUMINANCE,
IPP_VIDEO_CHROMINANCE,
IPP_VIDEO_ALPHA.
```

## Description

The functions `ippiQuantInvIntra_MPEG4_16s_C1I` and `ippiQuantInvInter_MPEG4_16s_C1I` are declared in the `ippvc.h` file. The functions `ippiQuantInvIntra_MPEG4_16s_I` and `ippiQuantInvInter_MPEG4_16s_I` are declared in the `ippalign.h` file. All these functions perform inverse quantization on intra/inter coded block, saturation and mismatch control (for the first inverse quantization method only) as it is specified in [ISO14496], subclause 7.4.4.

For `ippiQuantInvIntra_MPEG4_16s_C1I` and `ippiQuantInvInter_MPEG4_16s_C1I` output coefficients are saturated to lie in range  $[-2^{bitsPerPixel+3}, 2^{bitsPerPixel+3} - 1]$ .

These functions are used in the MPEG-4 encoder and decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

For `ippiQuantInvIntra_MPEG4_16s_I` and `ippiQuantInvInter_MPEG4_16s_I` output coefficients are saturated to lie in range  $[-2048, 2047]$ , that is, these functions support only  $bitsPerPixel = 8$ .

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one pointer is NULL.
<code>ippStsQPErr</code>	Indicates an error condition if $QP$ is out of the range $[1, 2^{bitsPerPixel-3} - 1]$ .

## VLC Decoding

---

### DecodeDCIntra\_MPEG4

*Decodes one DC coefficient for intra coded block.*

---

#### Syntax

```
IppStatus ippiDecodeDCIntra_MPEG4_1u16s(Ipp8u **ppBitStream, int *pBitOffset,  
    Ipp16s *pDC, int blockType);
```

#### Parameters

<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . The pointer is updated by the function.
<i>pDC</i>	Pointer to the output coefficient.
<i>blockType</i>	Indicates the block type, takes one of the following values: IPPVC_BLOCK_LUMA - for luma and alpha blocks, IPPVC_BLOCK_CHROMA - for chroma blocks.

#### Description

The function `ippiDecodeDCIntra_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding of the DC coefficient only for one intra coded block using VLC decoding process as specified in [\[ISO14496\]](#), subclause 7.4.1.1.

This function is used in the MPEG-4 decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is <code>NULL</code> .
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>pBitOffset</i> is out of the range <code>[0, 7]</code> .

`ippStsVLCCodeErr`

Indicates an error condition if an illegal code is detected through the DC decoding.

---

## DecodeCoeffsIntra\_MPEG4

*Decodes DCT coefficients for intra coded block.*

---

### Syntax

```
IppStatus ippiDecodeCoeffsIntra_MPEG4_1u16s(Ipp8u **ppBitStream, int
    *pBitOffset, Ipp16s *pCoeffs, int *pIndxLastNonZero, int rvlcFlag, int
    noDCFlag, int scan);
```

### Parameters

<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . The pointer is updated by the function.
<i>pCoeffs</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in it.
<i>rvlcFlag</i>	Flag, when set to '0' indicates that VLC tables B.16, B.18, B.19, and B.21 [ISO14496] are used in decoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24, and B.25 [ISO14496] are used.
<i>noDCFlag</i>	Flag, when set to '0' indicates that <i>pCoeffs</i> is set starting with zero element, otherwise - with the first element.
<i>scan</i>	Type of the scan, takes one of the following values:  IPPVC_SCAN_NONE, indicating that no inverse scan is performed, IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan. IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan,  See the corresponding enumerator on <a href="#">p.16-3</a> .

## Description

The function `ippiDecodeCoeffsIntra_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding of the DC coefficient (if `noDCFlag` is 0) and AC coefficients for one intra coded block using VLC decoding process as specified in [ISO14496], subclause 7.4.1. Additionally, the function can perform inverse scan.

This function is used in the MPEG-4 decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range [0, 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the decoding.

---

## DecodeCoeffsIntraRVLCBack\_MPEG4

*Decodes DCT coefficients in backward direction for intra coded block using RVLC.*

---

## Syntax

```
IppStatus ippiDecodeCoeffsIntraRVLCBack_MPEG4_1u16s(Ipp8u **ppBitStream, int
    *pBitOffset, Ipp16s *pCoeffs, int *pIndxLastNonZero, int noDCFlag);
```

## Parameters

<code>ppBitStream</code>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<code>pBitOffset</code>	Pointer to the bit position in the byte pointed by <code>*ppBitStream</code> . The pointer is updated by the function.
<code>pCoeffs</code>	Pointer to the output coefficients.

<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in it.
<i>noDCFlag</i>	Flag, when set to '0' indicates that <i>pCoeffs</i> is set starting with zero element, otherwise - with the first element.

## Description

The function `ippIDecodeCoeffsIntraRVLCBack_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs backward decoding of the DC coefficient (if *noDCFlag* is 0) and AC coefficients using variable length tables B.23, B.24, and B.25 [ISO14496] for one intra coded block. The backward RVLC decoding process is specified in [ISO14496], subclauses E.1.3 and E.1.4.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error when at least one input pointer is NULL.
<i>ippStsBitOffsetErr</i>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<i>ippStsVLCCodeErr</i>	Indicates an error condition if an illegal code is detected through the decoding.

---

## DecodeCoeffsInter\_MPEG4

*Decodes DCT coefficients for inter coded block.*

---

## Syntax

```
IppStatus ippIDecodeCoeffsInter_MPEG4_1u16s(Ipp8u **ppBitStream, int
    *pBitOffset, Ipp16s *pCoeffs, int *pIndxLastNonZero, int rvlcFlagint, int
    scan);
```

## Parameters

<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
--------------------	---

<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . The pointer is updated by the function.
<i>pCoeffs</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in it.
<i>rvlcFlag</i>	Flag, when set to '0' indicates that VLC tables B.17, B.18, B.19, and B.21 [ISO14496] are used in decoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24, and B.25 [ISO14496] are used.
<i>scan</i>	Type of the scan, takes one of the following values:  IPPVC_SCAN_NONE, indicating that no inverse scan is performed, IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan. IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan,  See the corresponding enumerator on <a href="#">p.16-3</a> .

## Description

The function `ippiDecodeCoefInter_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding of the DCT coefficients for one inter coded block. Inter DCT VLC decoding process is specified in [ISO14496], subclause 7.4.1. Additionally, the function can perform inverse scan.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the DCT decoding.

---

## DecodeCoeffsInterRVLCBack\_MPEG4

*Decodes DCT coefficients in backward direction for inter coded block using RVLC.*

---

### Syntax

```
IppStatus ippiDecodeCoeffsIntraRVLCBack_MPEG4_1u16s(Ipp8u **ppBitStream, int *pBitOffset, Ipp16s *pCoeffs, int *pIdxLastNonZero);
```

### Parameters

<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . The pointer is updated by the function.
<i>pCoeffs</i>	Pointer to the output coefficients.
<i>pIdxLastNonZero</i>	Pointer to the index of the last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in it.

### Description

The function `ippiDecodeCoeffsInterRVLCBack_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs backward decoding of the DC coefficient coefficients using variable length tables B.23, B.24 and B.25 [ISO14496] for one inter coded block. The backward RVLC decoding process is specified in [ISO14496], subclauses E.1.3 and E.1.4.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the decoding.

## ReconstructCoeffsInter\_MPEG4

*Decodes DCT coefficients, performs inverse scan and inverse quantization for inter coded block.*

---

### Syntax

```
IppStatus ippiReconstructCoeffsInter_MPEG4_1u16s(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pCoeffs, int* pIndxLastNonZero, int rvlcFlag, int scan,
    const IppiQuantInvInterSpec_MPEG4* pQuantInvInterSpec, int QP);
```

### Parameters

<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . The pointer is updated by the function.
<i>pCoeffs</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of last non-zero coefficient. In case of error during decoding, the index of the previous successfully decoded coefficient is stored in this parameter.
<i>rvlcFlag</i>	Flag, which when set to '0' indicates that VLC tables B.17, B.18, B.20 and B.22 <a href="#">[ISO14496]</a> are used when decoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24 and B.25 <a href="#">[ISO14496]</a> are used.
<i>scan</i>	Type of the inverse scan, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan.  See the corresponding enumerator on <a href="#">p.16-3</a> .
<i>pQuantInvInterSpec</i>	Pointer to the IppiQuantInvInterSpec_MPEG4 that was initialized by ippiQuantInvInterInit_MPEG4.
<i>QP</i>	Quantization parameter



## Description

The function `ippiReconstructCoeffsInter_MPEG4_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding, inverse scan and inverse quantization of the DCT coefficients for one inter coded block. Inter DCT VLC decoding process is specified in [ISO14496], subclause 7.4.1. Inverse scan process is specified in [ISO14496], subclause 7.4.2. Inverse quantization process is specified in [ISO14496], subclause 7.4.4. This function supports video data precision 4-12 bits per pixel, that is, after inverse quantization the coefficients are saturated to lie in the range:  $[-2^{bitsPerPixel+3}, 2^{bitsPerPixel+3} - 1]$ .

This function is used in the MPEG-4 decoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer (except <code>pQuantMatrix</code> ) is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range [0; 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the DCT decoding.
<code>ippStsQPErr</code>	Indicates an error condition if QP is out of the range $[1; 2^{bitsPerPixel-3} - 1]$ .

---

## DecodeVLCZigzag\_IntraDCVLC\_MPEG4, DecodeVLCZigzag\_IntraACVLC\_MPEG4

*Perform VLC decoding and inverse zigzag scan for one intra coded block.*

---

## Syntax

```
IppStatus ippiDecodeVLCZigzag_IntraDCVLC_MPEG4_1u16s( Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pDst, int predDir, IppVideoComponent videoComp);
IppStatus ippiDecodeVLCZigzag_IntraACVLC_MPEG4_1u16s(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pDst, int predDir);
```

## Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> will be updated after block decoding.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . Valid within the range 0 to 7. <i>pBitOffset</i> will be updated after block decoding.
<i>pDst</i>	Pointer to the quantized coefficient residuals (PQFs) of the current block.
<i>predDir</i>	AC prediction direction which is used to decide the zigzag scan pattern. It takes one of the following values:  IPP_VIDEO_NONE      AC prediction not used; perform classical zigzag scan;  IPP_VIDEO_HORIZONTAL   Horizontal prediction; perform alternate-vertical zigzag scan;  IPP_VIDEO_VERTICAL      Vertical prediction; perform alternate-horizontal zigzag scan.
<i>videoComp</i>	Video component type (luminance, chrominance or alpha) of the current block. It is used for DC coefficient decoding.

## Description

The functions `ippiDecodeVLCZigzag_IntraDCVLC_MPEG4_1u16s` and `ippiDecodeVLCZigzag_IntraACVLC_MPEG4_1u16s` are declared in the `ippalign.h` file. These functions perform VLC decoding and inverse zigzag scan for one intra coded block. Intra DC VLC decoding process is specified in [ISO14496], subclause 7.4.1.1. Intra AC VLC decoding process is specified in [ISO14496], subclauses 7.4.1.2, 7.4.1.3. Inverse zigzag scan is specified in [ISO14496], subclause 7.4.2.

The `IntraDCVLC` version uses Intra DC VLC to decode Intra DC coefficient, while the `IntraACVLC` version uses Intra AC VLC to decode Intra DC coefficient.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the pointers <i>ppBitStream</i> , <i>*ppBitStream</i> , <i>pBitOffset</i> , <i>pDst</i> is NULL.

---

<code>ippStsMP4BitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsMP4DCCodeErr</code>	(DC version only) Indicates an error condition if an illegal code is detected through the DC stream processing.
<code>ippStsMP4VLCCodeErr</code>	Indicates an error condition if an illegal Huffman code is detected through the VLC stream processing.
<code>ippStsMPEG4PredDirErr</code>	Indicates an error condition if <i>predDir</i> is not valid.

---

## DecodeVLCZigzag\_Inter\_MPEG4

*Performs VLC decoding and inverse zigzag scan for one inter coded block.*

---

### Syntax

```
IppStatus ippIDecodeVLCZigzag_Inter_MPEG4_1u16s(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pDst);
```

### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> will be updated after block decoding.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . Valid within the range 0 to 7. <i>pBitOffset</i> will be updated after block decoding.
<i>pDst</i>	Pointer to the quantized coefficient residuals (PQFs) of the current block.

### Description

The function `ippIDecodeVLCZigzag_Inter_MPEG4_1u16s` is declared in the `ippalign.h` file. This function performs VLC decoding for one inter coded block. Inter VLC decoding process is specified in [\[ISO14496\]](#), subclauses 7.4.1.2, 7.4.1.3. Inverse zigzag scan is specified in [\[ISO14496\]](#), subclause 7.4.2.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the pointers <code>ppBitStream</code> , <code>*ppBitStream</code> , <code>pBitOffset</code> , <code>pDst</code> is NULL.
<code>ippStsMP4BitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range [0, 7].
<code>ippStsMP4VLCCodeErr</code>	Indicates an error condition if an illegal Huffman code is detected through the VLC stream processing.

## Block Decoding

---

### DecodeBlockCoef\_Intra\_MPEG4

*Decodes the INTRA block coefficients.*

---

#### Syntax

```
IppStatus ippIDecodeBlockCoef_Intra_MPEG4_1u8u(Ipp8u** ppBitStream, int*
pBitOffset, Ipp8u* pDst, int step, Ipp16s* pCoefBufRow, Ipp16s* pCoefBufCol,
Ipp8u curQP, Ipp8u* pQPBuf, const Ipp8u* pQPMatrix, int blockIndex, int
intraDCVLC, int ACPredFlag);
```

#### Parameters

<code>ppBitStream</code>	Pointer to pointer to the current byte in the bitstream buffer, <code>*ppBitStream</code> will be updated after block decoding.
<code>pBitOffset</code>	Pointer to the bit position in the byte pointed by <code>ppBitStream</code> . Valid within the range 0 to 7. <code>pBitOffset</code> will be updated after block decoding.
<code>pDst</code>	Pointer to the block in the destination plane.
<code>step</code>	Width in bytes of the destination plane.
<code>pCoefBufRow</code>	Pointer to the coefficient row buffer, it will be updated after block decoding.
<code>pCoefBufCol</code>	Pointer to the coefficient column buffer, it will be updated after block decoding.

<i>curQP</i>	Quantization parameter of the macroblock which the current block belongs to.
<i>pQPBuf</i>	Pointer to the quantization parameter buffer.
<i>pQPMatrix</i>	Pointer to the quantization weighting matrix for intra macroblock. If it is NULL, it indicates to use the second inverse quantization method; otherwise, the first inverse quantization method is used.
<i>blockIndex</i>	Block index indicating the component type and position as defined in <a href="#">[ISO14496]</a> , subclause 6.1.3.8, Figure 6-5. Furthermore, index 6 to 9 indicates the alpha blocks spatially corresponding to luminance block 0 to 3 in the same macroblock.
<i>intraDCVLC</i>	A flag indicating if Intra DC VLC will be used to decode a DC coefficient instead of AC VLC. This flag is determined by <i>intra_dc_vlc_thr</i> and QP according to <a href="#">[ISO14496]</a> , Table 6-21.
<i>ACPredFlag</i>	A flag equal to <i>ac_pred_flag</i> (of luminance) or <i>ac_pred_flag_alpha</i> (of alpha block) indicating if the AC coefficients of the first row or first column should be predicted.

## Description

The function `ippiDecodeBlockCoef_Intra_MPEG4_1u8u` is declared in the `ippalign.h` file. This function decodes the INTRA block coefficients. It performs variable length decoding ([\[ISO14496\]](#), subclauses 7.4.1.1, 7.4.1.2, 7.4.1.3), inverse zigzag scan ([\[ISO14496\]](#)), subclause 7.4.2), inverse DC and AC prediction ([\[ISO14496\]](#)), subclause 7.4.3), inverse dequantization ([\[ISO14496\]](#), subclause 7.4.4) and inverse DCT. Output values are clipped to [0, 255] and are ready to be written to current frame within the destination plane.

The coefficient buffer should be updated according to the predefined structure. (See [Buffers](#) in this chapter for more details.)

This function should be used only when at least one non-zero AC coefficient of the current block exists in the bitstream.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the pointers <i>ppBitStream</i> , <i>*ppBitStream</i> , <i>pBitOffset</i> , <i>pDst</i> , <i>pQPBuf</i> , <i>pCoefBufRow</i> , <i>pCoefBufCol</i> is NULL.
<code>ippStsMP4BitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].

<code>ippStsStepErr</code>	Indicates an error condition if <i>step</i> is less than 8, or is not divisible by 4.
<code>ippStsMP4QPErr</code>	Indicates an error condition if <i>curQP</i> or necessary element of <i>pQPBuf</i> is out of the range [1, 31].
<code>ippStsMP4BlockIdxErr</code>	Indicates an error condition if <i>blockIndex</i> is out of the range [0, 9].
<code>ippStsMP4DCCodeErr</code>	Indicates an error condition if an illegal code is detected through the DC stream processing.
<code>ippStsMP4VLCCodeErr</code>	Indicates an error condition if an illegal Huffman code is detected through the VLC stream processing.

---

## DecodeBlockCoef\_Inter\_MPEG4

*Decodes the INTER block coefficients.*

---

### Syntax

```
IppStatus ippiDecodeBlockCoef_Inter_MPEG4_1u16s(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s* pDst, int QP, const Ipp8u* pQPMatrix);
```

### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer, <i>*ppBitStream</i> will be updated after block decoding.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . Valid within the range 0 to 7. <i>pBitOffset</i> will be updated after block decoding.
<i>pDst</i>	Pointer to the decoded residual buffer.
<i>QP</i>	Quantization parameter.
<i>pQPMatrix</i>	Pointer to the quantization weighting matrix for intra macroblock. If it is NULL, this indicates that the second inverse quantization method is to be used; otherwise, the first inverse quantization method is used.

## Description

The function `ippiDecodeBlockCoef_Inter_MPEG4_1u16s` is declared in the `ippalign.h` file. This function decodes the INTER block coefficients. This function performs variable length decoding ([ISO14496], subclauses 7.4.1.2, 7.4.1.3), inverse classical zigzag scan ([ISO14496], subclause 7.4.2), inverse dequantization ([ISO14496], subclause 7.4.4) and inverse DCT. The output buffer contains the residuals for further reconstruction.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the pointers <code>ppBitStream</code> , <code>*ppBitStream</code> , <code>pBitOffset</code> , <code>pDst</code> is NULL.
<code>ippStsMP4BitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range [0, 7].
<code>ippStsMP4QPErr</code>	Indicates an error condition if <code>QP</code> is out of the range [1, 31].
<code>ippStsMP4VLCCodeErr</code>	Indicates an error condition if an illegal Huffman code is detected through the VLC stream processing.

## Postprocessing

---

## FilterDeblocking8x8HorEdge\_MPEG4, FilterDeblocking8x8VerEdge\_MPEG4

*Perform deblocking filtering on a horizontal or vertical edge of two adjacent blocks.*

---

## Syntax

```
IppStatus ippiFilterDeblocking8x8HorEdge_MPEG4_8u_C1IR(Ipp8u* pSrcDst, int
    step, int QP, int THR1, int THR2);

IppStatus ippiFilterDeblocking8x8VerEdge_MPEG4_8u_C1IR(Ipp8u* pSrcDst, int
    step, int QP, int THR1, int THR2);
```

### Parameters

<i>pSrcDst</i>	Pointer to the first pixel of lower (HorEdge) or right (VerEdge) block.
<i>step</i>	Width in bytes of the source plane.
<i>QP</i>	Quantization parameter.
<i>THR1</i> , <i>THR2</i>	Threshold values that specify the filter mode ( <a href="#">[ISO14496]</a> Annex F.3.1).

### Description

The functions `ippiFilterDeblocking8x8HorEdge_MPEG4_8u_C1IR` and `ippiFilterDeblocking8x8VerEdge_MPEG4_8u_C1IR` are declared in the `ippvc.h` file. These functions perform deblocking filtering of two adjacent blocks on horizontal and vertical edges, respectively ([\[ISO14496\]](#), Annex F.3.1).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the pointer <i>pSrcDst</i> is NULL.
<code>ippStsMP4QPErr</code>	Indicates an error condition if <i>QP</i> is out of range [1, 31]

---

## FilterDeringingThreshold\_MPEG4

*Computes threshold values for the deringing filtering through a macroblock.*

---

### Syntax

```
IppStatus ippiFilterDeringingThreshold_MPEG4_8u_P3R(Ipp8u* pSrcY, int stepY,
const Ipp8u* pSrcCb, int stepCb, const Ipp8u* pSrcCr, int stepCr, int
threshold[6]);
```

### Parameters

<i>pSrcY</i>	Pointer to the left upper Y-block in the current macroblock.
<i>stepY</i>	Width in bytes of the luminance plane.
<i>pSrcCb</i>	Pointer to the Cb-block in the current macroblock.
<i>stepCb</i>	Width in bytes of the chrominance (Cb) plane.



---

<i>pSrcCr</i>	Pointer to the Cr-block in the current macroblock.
<i>stepCr</i>	Width in bytes of the chrominance (Cr) plane.
<i>threshold</i>	Array of the threshold values for all blocks.

### Description

The function `ippiFilterDeringingThreshold_MPEG4_8u` is declared in the `ippvc.h` file. This function performs threshold determination ([[ISO14496](#)], Annex F.3.2.1), which is the first subprocess of the deringing filtering. The obtained threshold values are required for the function [FilterDeringingSmooth8x8\\_MPEG4](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.

---

## FilterDeringingSmooth8x8\_MPEG4

*Performs deringing filtering of a block.*

---

### Syntax

```
IppStatus ippiFilterDeringingSmooth8x8_MPEG4_8u_C1R(Ipp8u* pSrc, int srcStep,  
Ipp8u* pDst, int dstStep, int QP, int threshold);
```

### Parameters

<i>pSrc</i>	Pointer to the source block.
<i>srcStep</i>	Width in bytes of the source plane.
<i>QP</i>	Quantization parameter ( <code>quantiser_scale</code> ).
<i>threshold</i>	Threshold value for the block.
<i>pDst</i>	Pointer to the destination block.
<i>dstStep</i>	Width in bytes of the destination plane.

## Description

The function `ippiFilterDeringingSmooth8x8_MPEG4_8u` is declared in the `ippvc.h` file. This function performs deringing filtering, specifically, index acquisition and adaptive smoothing ([ISO14496], Annex F.3.2.1, 3.2.2) of the source block *pSrc*, and stores the result in the destination block *pDst*. The threshold value *threshold* is returned by the auxiliary function [FilterDeringingThreshold\\_MPEG4](#) that should be called beforehand.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if one of the specified pointers is NULL.
<code>ippStsMP4QPErr</code>	Indicates an error condition if <i>QP</i> is out of range [1, 31]

## Shape Decoding

---

# DecodeCAEIntraH\_MPEG4, DecodeCAEIntraV\_MPEG4

*Perform Context Arithmetic Code decoding in the intra macroblock.*

---

## Syntax

```
IppStatus ippiDecodeCAEIntraH_MPEG4_1u8u(Ipp8u** ppBitStream, int* pBitOffset,  
    Ipp8u* pBinarySrcDst, int step, int blockSize);  
IppStatus ippiDecodeCAEIntraV_MPEG4_1u8u(Ipp8u** ppBitStream, int* pBitOffset,  
    Ipp8u* pBinarySrcDst, int step, int blockSize);
```

## Parameters

<i>ppBitStream</i>	Pointer to the pointer to the current byte from which the intra block starts.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> will be updated after block decoding.

<i>pBinarySrcDst</i>	Pointer to the current block of binary value pixels. The left and the top borders should be loaded beforehand.
<i>step</i>	Width in bytes of the source-destination binary plane.
<i>blockSize</i>	Size of the block that may take values 16, 8, 4.

### Description

The functions `ippiDecodeCAEIntraH_MPEG4_1u8u` and `ippiDecodeCAEIntraV_MPEG4_1u8u` are declared in the `ippalign.h` file. These functions perform Context Arithmetic Code decoding in intra macroblock. H indicates that the scan type is horizontal. V indicates that the scan type is vertical. Convert ratio is supported in these functions.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

---

## DecodeCAEInterH\_MPEG4, DecodeCAEInterV\_MPEG4

*Perform Context Arithmetic Code decoding in the inter macroblock.*

---

### Syntax

```
IppStatus ippiDecodeCAEInterH_MPEG4_1u8u(Ipp8u** ppBitStream, int* pBitOffset,
    const Ipp8u* pBinarySrcPred, int offsetPred, Ipp8u* pBinarySrcDst, int step,
    int blockSize);

IppStatus ippiDecodeCAEInterV_MPEG4_1u8u(Ipp8u** ppBitStream, int* pBitOffset,
    const Ipp8u* pBinarySrcPred, int offsetPred, Ipp8u* pBinarySrcDst, int step,
    int blockSize);
```

### Parameters

<i>ppBitStream</i>	Pointer to the pointer to the current byte from which the inter block starts.
--------------------	---

<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> will be updated after block decoding.
<i>pBinarySrcPred</i>	Pointer to the related macroblock in the reference binary plan. The left and top borders should be loaded beforehand. But the pointer points to the top-left corner of this macroblock, not the extended zone.
<i>offsetPred</i>	Bit position of the first pixel in the reference macroblock, valid within the range 0 to 7, where: — most significant bit = zero (0) — least significant bit = seven (7). Bit positions in a byte vary from 0 to 7.
<i>pBinarySrcDst</i>	Pointer to the current block of binary value pixels. The left and the top border should be loaded beforehand.
<i>step</i>	Width in bytes of the source-destination binary plane. If the blocksize is not equal to 16, the argument indicates binary buffer step.
<i>blockSize</i>	Size of the block that may take values 16, 8, 4.

## Description

The functions `ippiDecodeCAEInterH_MPEG4_1u8u` and `ippiDecodeCAEInterV_MPEG4_1u8u` are declared in the `ippalign.h` file. These functions perform Context Arithmetic Code decoding in inter macroblock. H indicates that the scan type is horizontal. V indicates that the scan type is vertical. Convert ratio is supported in these functions.

Reference binary plane and current binary plane have the same step. If the blocksize is not equal to 16 (convert ratio takes effect), *step* indicates both the reference binary buffer and the current binary buffer.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

---

## DecodeMVS\_MPEG4

*Decodes motion vectors of shape according to specification.*

---

### Syntax

```
IppStatus ippiDecodeMVS_MPEG4 (Ipp8u** ppBitStream, int* pBitOffset,
    IppMotionVector* pSrcDstMVS, const Ipp8u* pSrcBABMode, int stepBABMode,
    const IppMotionVector* pSrcMVLeftMB, const IppMotionVector* pSrcMVUpperMB,
    const IppMotionVector* pSrcMVUpperRightMB, const Ipp8u* pTranspLeftMB, const
    Ipp8u* pTranspUpperMB, const Ipp8u* pTranspUpperRightMB, int stepBM, int
    predFlag);
```

### Parameters

<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> will be updated after block decoding.
<i>pSrcDstMVS</i>	Pointer to the shape motion vector buffer of the current binary alpha block (BAB).
<i>pSrcBABMode</i>	Pointer to the BAB mode buffer of the current BAB, which is stored in the BAB mode plane.
<i>pSrcMVLeftMB</i>	Pointer to the motion vector buffer of the macroblocks at the left side of the current macroblock.
<i>pSrcMVUpperMB</i>	Pointer to the motion vector buffer of the macroblocks at the upper side of the current macroblock.
<i>pSrcMVUpperRightMB</i>	Pointer to the motion vector buffer of the macroblocks at the upper-right side of the current macroblock.
<i>pTranspLeftMB</i>	Pointer to the transparent vector buffer of the macroblocks at the left side of the current macroblock.
<i>pTranspUpperMB</i>	Pointer to the transparent vector buffer of the macroblocks at the upper side of the current macroblock.
<i>pTranspUpperRightMB</i>	Pointer to the transparent vector buffer of the macroblocks at the upper-right side of the current macroblock.

<i>stepBM</i>	Width of the BAB mode plane. This value cannot be less than zero.
<i>predFlag</i>	Flag that is set to zero if the current video object plane (VOP) is a bidirectionally predictive-coded video object plane (BVOP) or the current video object layer (VOL) is in shape only mode; else, the flag is nonzero.

### Description

The function `ippiDecodeMVS_MPEG4` is declared in the `ippalign.h` file. This function decodes motion vectors for the binary alpha block.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

---

## PadMBPartial\_MPEG4

*Performs general padding if the current macroblock is partial.*

---

### Syntax

```
IppStatus ippiPadMBPartial_MPEG4_8u_P4R(const Ipp8u* pSrcBAB, const Ipp32u*
pSrcTrasptMBLeft, Ipp8u* pSrcDstCurrY, Ipp8u* pSrcDstCurrCb, Ipp8u*
pSrcDstCurrCr, Ipp8u* pSrcDstCurrA, Ipp8u* pSrcDstPadded, int iMBX, int
iMBY, int stepYA, int stepCbCr, int stepBinary);
```

### Parameters

<i>pSrcBAB</i>	Pointer to the binary alpha block in current VOP.
<i>pSrcTrasptMBLeft</i>	Pointer to transparent buffer of the left neighboring macroblock.
<i>pSrcDstCurrY</i>	Pointer to top-left of the current luminance block in current VOP.
<i>pSrcDstCurrCb</i>	Pointer to top-left of the current chrominance (Cb) block in current VOP.
<i>pSrcDstCurrCr</i>	Pointer to top-left of the current chrominance (Cr) block in current VOP.
<i>pSrcDstCurrA</i>	Pointer to top-left of the current alpha block in current VOP.

---

<i>pSrcDstPadded</i>	Pointer to the padded buffer, which indicates whether the related macroblock is padded or not. It makes sense when related macroblock is transparent.
<i>iMBX</i>	Current macroblock X direction index in VOP, the start one is 0. This argument is equal or greater than 0.
<i>iMBY</i>	Current macroblock Y direction index in VOP, the start one is 0. This argument is equal or greater than 0.
<i>stepYA</i>	Width in bytes of luminance/alpha plane. This argument is equal or greater than 16 and multiple of 8.
<i>stepCbCr</i>	Width in bytes of chrominance plane. This argument is equal or greater than 8 and multiple of 8.
<i>stepBinary</i>	Width in bytes of binary plane. This argument is equal or greater than 2.

### Description

The function `ippiPadMBPartial_MPEG4_8u_P4R` is declared in the `ippalign.h` file. This function pads a current partially filled macroblock. It also pads the neighboring macroblocks.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

---

## PadMBTransparent\_MPEG4

*Performs general padding if the current macroblock is transparent.*

---

### Syntax

```
IppStatus ippiPadMBTransparent_MPEG4_8u_P4R(const Ipp32s* pSrcTrasptMBLeft,
Ipp8u* pSrcDstCurrY, Ipp8u* pSrcDstCurrCb, Ipp8u* pSrcDstCurrCr, Ipp8u*
pSrcDstCurrA, Ipp8u* pSrcDstPadded, Ipp8u grayVal, int iMBX, int iMBY, int
iMBXLimit, int iMBYLimit, int stepYA, int stepCbCr);
```

### Parameters

<i>pSrcTrasptMBLeft</i>	Pointer to transparent buffer of the left neighboring macroblock.
-------------------------	---

<i>pSrcDstCurrY</i>	Pointer to top-left of the current luminance block.
<i>pSrcDstCurrCb</i>	Pointer to top-left of the current chrominance (Cb) block.
<i>pSrcDstCurrCr</i>	Pointer to top-left of the current chrominance (Cr) block.
<i>pSrcDstCurrA</i>	Pointer to top-left of the current alpha block in current VOP.
<i>pSrcDstPadded</i>	Pointer to the padded buffer, which indicates whether the related macroblock is padded or not.
<i>grayVal</i>	Gray value to fill the exterior macroblock/block. It should be set to $2^{\text{bits\_per\_pixel} - 1}$ , where $\text{bits\_per\_pixel} \leq 8$ .
<i>imBX</i>	Current macroblock X direction index in VOP, the start one is 0. This argument is equal or greater than 0 and equal or less than <i>imBXLimit</i> .
<i>imBY</i>	Current macroblock Y direction index in VOP, the start one is 0. This argument is equal or greater than 0 and equal or less than <i>imBYLimit</i> .
<i>imBXLimit</i>	The number of macroblocks in the current VOP X direction. This argument is greater than 0.
<i>imBYLimit</i>	The number of macroblocks in the current VOP Y direction. This argument is greater than 0.
<i>stepYA</i>	Width in bytes of luminance/alpha plane. This argument is equal or greater than 16 and multiple of 8.
<i>stepCbCr</i>	Width in bytes of chrominance plane. This argument is equal or greater than 8 and multiple of 8.

## Description

The function `ippiPadMBTransparent_MPEG4_8u_P4R` is declared in the `ippalign.h` file. This function pads a current transparent macroblock. It also pads the neighboring macroblocks if such padding is needed.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.



---

## PadMBOpaque\_MPEG4

*Performs general padding if the current macroblock is opaque.*

---

### Syntax

```
IppStatus ippPadMBOpaque_MPEG4_8u_P4R(const Ipp32s* pSrcTrasptMBLeft, Ipp8u*
    pSrcDstCurrY, Ipp8u* pSrcDstCurrCb, Ipp8u* pSrcDstCurrCr, Ipp8u*
    pSrcDstCurrA, Ipp8u* pSrcDstPadded, int iMBX, int iMBY, int stepYA, int
    stepCbCr);
```

### Parameters

<i>pSrcTrasptMBLeft</i>	Pointer to transparent buffer of the left neighboring macroblock.
<i>pSrcDstCurrY</i>	Pointer to top-left of the current luminance block in current VOP.
<i>pSrcDstCurrCb</i>	Pointer to top-left of the current chrominance (Cb) block in current VOP.
<i>pSrcDstCurrCr</i>	Pointer to top-left of the current chrominance (Cr) block in current VOP.
<i>pSrcDstCurrA</i>	Pointer to top-left of the current alpha block in current VOP.
<i>pSrcDstPadded</i>	Pointer to the related padded buffer.
<i>iMBX</i>	Current macroblock X direction index in VOP, the start one is 0. This argument is equal or greater than 0.
<i>iMBY</i>	Current macroblock Y direction index in VOP, the start one is 0. This argument is equal or greater than 0.
<i>stepYA</i>	Width in bytes of luminance/alpha plane. This argument is equal or greater than 16 and multiple of 8.
<i>stepCbCr</i>	Width in bytes of chrominance plane. This argument is equal or greater than 8 and multiple of 8.

### Description

The function `ippPadMBOpaque_MPEG4_8u_P4R` is declared in the `ippalign.h` file. This function pads a current opaque macroblock. It also pads the neighboring macroblocks if such padding is needed.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

## MPEG-4 Video Encoder Functions

This section describes Intel IPP functions that are built to support the ISO/IEC 14496-2 MPEG-4 video encoder. MPEG-4 ([\[ISO14496\]](#)) is a widely used coding method for video signals in various applications such as digital storage media, internet, various forms of wired or wireless communications, etc. The functions cover the following aspects of MPEG-4 encoder:

- progressive, non-scalable texture encoding
- block-based VLC encoding and zigzag scan
- motion vector encoding
- motion estimation, including modified full length search (SEA, successive elimination algorithm) and fast search (MVFAST, motion vector field adaptive search technique), including integer pixel search and half pixel search, including 16x16 search and 8x8 search
- block layer coefficient encoding, including intra-DC/AC prediction (for intra blocks), quantization, and DCT, with appropriate clipping on each step, also provides reconstructed data
- rate control part, which support VOP based solution.

The remainder of the section provides description of the IPP MPEG-4 encoder functions usage, macro/data structures definition and the detailed descriptions of individual encoder functions.

### Data Types and Structures

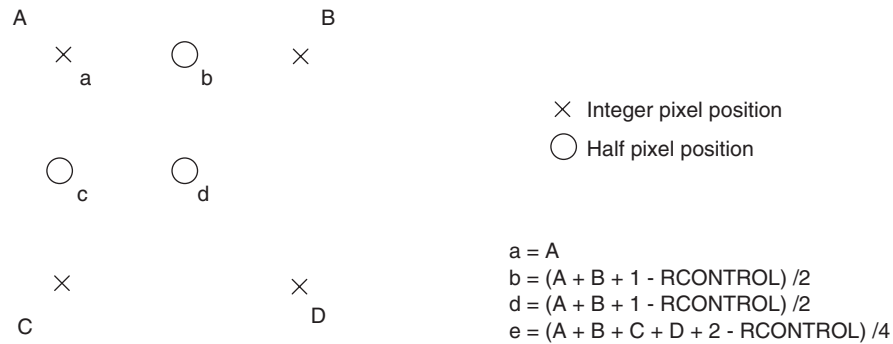
The basic data types and structures for MPEG-4 encoder are the same as described above for the IPP MPEG-4 decoder. See the corresponding subsections [Video Components](#), [Pixel Planes and Alpha Plane](#), [Macroblock Types](#), [Motion Vector](#), [Transparent Status](#), [Direction](#), [Rectangle Plane](#), and [Buffers](#).

Two kinds of motion vectors are used in the Intel IPP MPEG-4 codec. One is for texture (in Q1 format) and the other is for shape (in Q0 format).

IPP MPEG4 encoder uses Bilinear Interpolation type for motion estimation, compensation, and reconstruction.

```
enum {
    IPP_VIDEO_INTEGER_PIXEL= 0, /* case "a" in Figure 16-40 */
    IPP_VIDEO_HALF_PIXEL_X= 1, /* case "b" in Figure 16-40 */
    IPP_VIDEO_HALF_PIXEL_Y= 2, /* case "c" in Figure 16-40 */
    IPP_VIDEO_HALF_PIXEL_XY= 3 /* case "d" in Figure 16-40 */
};
```

**Figure 16-40 Halfpixel Prediction by Bilinear Interpolation**



A9152-01

## MVFAST buffer

Allocate two buffers storing the search status for the 16\*16 integer pixel search and the 8\*8 integer pixel search respectively. For each buffer, there is one bit for each searching point. A bit value of 1 indicates that the pointer has been checked before or that it is out of search range. Any other value indicates that this candidate should be checked. Accordingly, the search range determines the buffer size. The buffer size is as follows:

```
Buffer size = (2*searchRange+5) * ((searchRange+1)/8+searchRange/8
+ 4)
(Unit: byte)
```



The value  $v$  represented by the word is determined as follows (similar to IEEE double precision standard):

- “”)
  - If  $E = 231-1$  and  $F$  is zero and  $S$  is 1, then  $v = -\text{Infinity}$ .
  - If  $E = 231-1$  and  $F$  is zero and  $S$  is 0, then  $v = \text{Infinity}$ .
  - If  $0 < E < 231-1$  then  $v = (-1)^S * 2^{(E-230)} * (1.F)$  where “1.F” is intended to represent the binary number created by prefixing  $F$  with an implicit leading 1 and a binary point.
  - If  $E = 0$  and  $F$  is zero and  $S$  is 1, then  $v = -0$ .
  - If  $E = 0$  and  $F$  is zero and  $S$  is 0, then  $v = 0$ .

Parameters for MPEG4 rate control are packed into a data structure called `IppiSTATRC`.

```
typedef struct
{
    Ipp32uc x1;    /* the first order coefficients */
    Ipp32uc x2;    /* the second order coefficients */
    Ipp32u      rs; /* bit rate for sequence. for example, 24000
bits/sec */
    Ipp32u      rf; /* bits used for the first frame, for example, 10000
bits */
    Ipp32u      rc; /* bits used for the current frame after encoding
*/
    Ipp32u      rp; /* bits to be removed from the buffer per picture
*/
    int         ts; /* number of seconds for the sequence, for
example, 10 sec*/
    Ipp32uc ec;    /* mean absolute difference for the current
frame
                    after motion compensation */
    Ipp32uc ep;    /* mean absolute difference for the previous
frame
                    after motion compensation */
    Ipp32u qc; /* quantization level used for the current frame */
    Ipp32u qp; /* quantization level used for the previous frame */
}
```

```

    Ipp32u nr; /* number of P frames remaining for encoding */
    Ipp32u nc; /* number of P frames coded */
    Ipp32u ns; /* distance between encoded frames */
    int rr;      /* number of bits remaining for encoding this
sequence */
    Ipp32u t;    /* target bit to be used for the current frame */
    Ipp32u s;    /* number of bits used for encoding the previous
frame */
    Ipp32u hc;   /* header and motion vector bits used in the current
frame */
    Ipp32u hp;   /* header and motion vector bits used in the previous
frame */
    Ipp32u bs;   /* buffer size */
    int bl;      /* current buffer level */
    int         skipNextFrame; /* TRUE if buffer is full */
    Ipp32u wQp[RC_MAX_SLIDING_WINDOW];
                /* quantization levels for the past frames */
    Ipp32uc wRp[RC_MAX_SLIDING_WINDOW];
                /* scaled encoding complexity used for the past
frames */
    int wRejected[RC_MAX_SLIDING_WINDOW]; /* outliers */
    int vopArea;
}IppiSTATRC;

```

## Motion Estimation

---

## SumNorm\_VOP\_MPEG4

*Performs summation of a block of indicated size.*

---

### Syntax

```

IppStatus ippiSumNorm_VOP_MPEG4_8u16u(Ipp8u* pSrcRef, IppiRect* pSrcRefRect,
    Ipp16u* pDstSumRef, int flag, int step);

```

### Parameters

<i>pSrcRef</i>	Pointer to the original or reconstructed reference Y-plane, the pointer position is the top left of the padded plane.
<i>pSrcRefRect</i>	Pointer to the valid rectangular in reference plane, which describes the target summation zone.
<i>pDstSumRef</i>	Pointer to the summation plane, the pointer position is the top left of the padded plane.
<i>flag</i>	Algorithm selected flag, indicates the size of sub-block here.
<i>step</i>	The step in reference Y-plane, and reference summation plane.

### Description

The function `ippiSumNorm_VOP_MPEG4_8u16u` is declared in the `ippalign.h` file. This function performs summation of a block of indicated size, as required for Successive Elimination Algorithm (SEA) motion estimation.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

## BlockMatch\_Integer\_16x16\_SEA

*Performs 16x16 size block match with sub-region.*

### Syntax

```
IppStatus ippiBlockMatch_Integer_16x16_SEA (Ipp8u * pSrcRef, Ipp8u * pSrcCurr,
      Ipp16u * pSrcSumBlk, IppMotionVector * pSrcRefMV, IppCoordinate *
      pSrcPointPos, IppiRect * pSrcRefRect, int * pSrcDstminSAD, IppMotionVector *
      pDstMV, int step, int searchRange, int flag);
```

### Parameters

<i>pSrcRef</i>	Pointer to the original or reconstructed reference Y-plane, the pointer position is the same as the current macroblock's position in the current plane.
----------------	---

<i>pSrcCurr</i>	Pointer to the current original macroblock, which has been extracted from the current original plane.
<i>pSrcSumBlk</i>	Pointer to the current macroblock in the summation plane.
<i>pSrcRefMV</i>	Pointer to the predicted motion vector.
<i>pSrcPointPos</i>	Pointer to the position of the current macroblock in the current plane.
<i>pSrcRefRect</i>	Pointer to the valid rectangular in reference plane.
<i>pSrcDstminSAD</i>	Pointer to the minSAD, which is from SAD16X16 at mv = (0, 0).
<i>pDstMV</i>	Pointer to the destination motion vector for the current macroblock with integer pixel definition.
<i>step</i>	The step in reference Y-plane, and reference summation plane.
<i>searchRange</i>	Search range in the 16X16 integer pixel search.
<i>flag</i>	Algorithm selected flag, indicates the size of the sub-block in SEA (8, 4).

## Description

The function `ippiBlockMatch_Integer_16x16_SEA` is declared in the `ippalign.h` file. This function performs a 16x16 size block match with a sub-region, using SEA.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.




---

**NOTE.** The step for reference Y-plane and summation Y-plane is the same, but the current macroblock is stored contiguously.

---



---

## MotionEstimation\_16x16\_SEA

*Completes 16X16 size motion estimation using core SEA.*

---

### Syntax

```
IppStatus ippIMotionEstimation_16x16_SEA (Ipp8u * pSrcRef, Ipp8u *
    pSrcReconRef, Ipp16u *pSrcSumBlk, Ipp8u * pSrcCurr, IppiRect * pSrcRefRect,
    IppCoordinate * pSrcPointPos, IppMotionVector *pSrcRefMV, IppMotionVector *
    pDstMV, Ipp8u* pDstPreMbtype, int* pDstSAD, int step, int roundControl, int
    searchRange, int flag);
```

### Parameters

<i>pSrcRef</i>	Pointer to the original <i>Y</i> -plane, the pointer position is the same as the current macroblock's position in the current plane.
<i>pSrcReconRef</i>	Pointer to the reconstructed reference <i>Y</i> -plane, the pointer position is the same as the current macroblock's position in the current plane.
<i>pSrcSumBlk</i>	Pointer to the current macroblock in the summation plane.
<i>pSrcCurr</i>	Pointer to the current original macroblock, which has been extracted from the current original plane.
<i>pSrcRefRect</i>	Pointer to the valid rectangular in reference plane.
<i>pSrcPointPos</i>	Pointer to the position of the current macroblock in the current plane.
<i>pSrcRefMV</i>	Pointer to the predicted motion vector generated from the neighboring motion vector.
<i>pDstMV</i>	Pointer to the destination 4-motion vectors.
<i>pDstPreMbtype</i>	Pointer to pre-Mbtype, which stores the Intra/Inter, 1MV/4MV information.
<i>pDstSAD</i>	Pointer to the least SAD after motion estimation.
<i>step</i>	The step in reference <i>Y</i> -plane.
<i>roundControl</i>	Rounding control bit for half pixel motion estimation.
<i>searchRange</i>	Search range in the 16X16 integer pixel search.
<i>flag</i>	Size of the sub-block for SEA.

## Description

The function `ippiMotionEstimation_16x16_SEA` is declared in the `ippalign.h` file. This function completes the 16X16 size motion estimation using the core SEA. The function covers the 16X16 integer/half pixel search, 8X8 integer and half pixel search, and also decides the intra/inter choice and 1MV/4MV choice. At the same time, the function provides the summation of current MB's residual, which is required for the rate control module.

The pointer `pDstMV` points to the first vector in the 4 MV buffer, 4 MVs are stored continuously. If 1MV mode is selected, then 4 MV buffer stores the same vector.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.




---

**NOTE.** The step for reference Y-plane and summation Y-plane is the same, but the current macroblock is stored contiguously.

---



---

## BlockMatch\_Integer\_16x16\_MVFAST

*Performs 16x16 size block match with large and/or small diamond search.*

---

## Syntax

```
IppStatus ippiBlockMatch_Integer_16x16_MVFAST (Ipp8u * pSrcRef, Ipp8u *
    pSrcCurr, IppMotionVector *pSrcCanMV, IppMotionVector *pSrcRefMV,
    IppCoordinate * pSrcPointPos, IppiRect * pSrcRefRect, Ipp8u * pSrcSadMap,
    int * pFlag, int * pSrcDstSAD, IppMotionVector * pDstMV, int refStep, int
    searchRange);
```

## Parameters

<code>pSrcRef</code>	Pointer to the original or reconstructed reference Y-plane, the pointer position is the same as the current macroblock's position in the current plane.
----------------------	---

---

<i>pSrcCurr</i>	Pointer to the current original macroblock, which has been extracted from the current original plane.
<i>pSrcCanMV</i>	Pointer to the left, top and right-top reference motion vector respectively.
<i>pSrcRefMV</i>	Pointer to the predicted motion vector.
<i>pSrcPointPos</i>	Pointer to the position of the current macroblock in the current plane.
<i>pSrcRefRect</i>	Pointer to the valid rectangular in reference plane.
<i>pSrcSadMap</i>	Pointer to the initial address of the bit plane, which is used to store the state of each search point.
<i>pFlag</i>	Pointer to the flag used for SAD calculation.
<i>pSrcDstSAD</i>	Pointer to the initial SAD. As an output argument points to the updated SAD
<i>pDstMV</i>	Pointer to the destination motion vector for the current macroblock with integer pixel definition.
<i>refStep</i>	Step in reference Y-plane.
<i>searchRange</i>	Search range in the 16X16 integer pixel search.

## Description

The function `ippiBlockMatch_Integer_16x16_MVFAST` is declared in the `ippalign.h` file. This function performs 16x16 size block match with large and/or small diamond search. The buffer size for *pSrcSadMap* is  $(2 * \text{SearchRange} + 5) * ((\text{SearchRange} + 1) / 8 + \text{SearchRange} / 8 + 4)$  bytes.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

---

## MotionEstimation\_16x16\_MVFAST

*Performs fast motion estimation using the MVFAST algorithm.*

---

### Syntax

```
IppStatus ippiMotionEstimation_16x16_MVFAST (Ipp8u * pSrcRef, Ipp8u *  
    pSrcReconRef, Ipp8u * pSrcCurr, IppMotionVector *pSrcCanMV, IppMotionVector  
    *pSrcRefMV, IppCoordinate * pSrcPointPos, IppiRect * pSrcRefRect, Ipp8u *  
    pSrcSADMap, Ipp8u * pSrcBlockSADMap, IppMotionVector * pDstMV, Ipp8u  
    *pDstPreMbtype, int* pDstSAD, int step, int roundControl, int searchRange);
```

### Parameters

<i>pSrcRef</i>	Pointer to the original <i>Y</i> -plane, the pointer position is the same as the current macroblock's position in the current plane.
<i>pSrcReconRef</i>	Pointer to the reconstructed reference <i>Y</i> -plane, the pointer position is the same as the current macroblock's position in the current plane.
<i>pSrcCurr</i>	Pointer to the current original macroblock, which has been stored in a 16X16 size buffer.
<i>pSrcCanMV</i>	Pointer to the left, top, and right-top reference motion vector respectively.
<i>pSrcRefMV</i>	Pointer to the predicted motion vector.
<i>pSrcPointPos</i>	Pointer to the position of the current macroblock in the current plane.
<i>pSrcRefRect</i>	Pointer to the valid rectangular in reference plane.
<i>pSrcSADMap</i>	Pointer to the initial address of the bit plane in 16*16 block match.
<i>pSrcBlockSADMap</i>	Pointer to the initial address of the bit plane in 8*8 block match.
<i>pDstMV</i>	Pointer to the destination 4-motion vectors.
<i>pPreDstMbtype</i>	Pointer to the macroblock type: Inter1v, Inter4v or Intra.
<i>pDstSAD</i>	Pointer to the least SAD after motion estimation.
<i>step</i>	Step in reference <i>Y</i> -plane.
<i>roundControl</i>	Rounding control bit for half pixel motion estimation.
<i>searchRange</i>	Search range in the 16X16 integer pixel search.

## Description

The function `ippiMotionEstimation_16x16_MVFAST` is declared in the `ippalign.h` file. This function performs fast motion estimation using the MVFAST algorithm. Refer to N3675(Motion Vector Field-adaptive Search Algorithm).

The pointer `pSrcCurr` points to a continuous macroblock size buffer, which stores the extracted current original pixel from the current original Y-plane.

The pointer `pDstMV` points to the starting address of the motion vector buffer. If 4MV mode is selected, this buffer can store 4-motion vectors. If 1MV mode is selected, the single motion vector is duplicated into 4 copies, then stored in this motion vector buffer.

For the buffer size and initialization for `pSrcSadMap` and `pSrcBlockSadMap`, see [Figure 16-41](#) and its context.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

---

## ComputeTextureErrorBlock\_SAD

*Computes texture error of the block with SAD exported.*

---

### Syntax

```
IppStatus ippiComputeTextureErrorBlock_SAD_8u16s(const Ipp8u *pSrc, int
srcStep, const Ipp8u *pSrcRef, Ipp16s * pDst, int *pDstSAD);
```

### Parameters

<code>pSrc</code>	Pointer to the source plane. This pointer should be aligned on an 8-byte boundary.
<code>srcStep</code>	Step of the source plane.
<code>pSrcRef</code>	Pointer to the reference buffer, an 8x8 block. This pointer should be aligned on an 8-byte boundary.
<code>pDst</code>	Pointer to the destination buffer, an 8x8 block. This pointer should be aligned on an 8-byte boundary.

*pDstSAD* Pointer to the SAD value.

### Description

The function `ippiComputeTextureErrorBlock_SAD_8u16s` is declared in the `ippalign.h` file. This function computes texture error of the block. SAD is also exported.

### Return Values

`ippStsNoErr` Indicates no error.

`ippStsBadArgErr` Indicates bad arguments.

---

## ComputeTextureErrorBlock

*Computes texture error of the block.*

---

### Syntax

```
IppStatus ippiComputeTextureErrorBlock_8u16s(const Ipp8u *pSrc, int srcStep,
      const Ipp8u *pSrcRef, Ipp16s *pDst);
```

### Parameters

*pSrc* Pointer to the source plane. This pointer should be aligned on an 8-byte boundary.

*srcStep* Step of the source plane.

*pSrcRef* Pointer to the reference buffer, an 8x8 block. This pointer should be aligned on an 8-byte boundary.

*pDst* Pointer to the destination buffer, an 8x8 block. This pointer should be aligned on an 8-byte boundary.

### Description

The function `ippiComputeTextureErrorBlock_8u16s` is declared in the `ippalign.h` file. This function computes the texture error of the block.

### Return Values

`ippStsNoErr` Indicates no error.

`ippStsBadArgErr` Indicates bad arguments.

## Quantization

---

### QuantIntraInit\_MPEG4, QuantInterInit\_MPEG4

*Initialize specification structures.*

---

#### Syntax

```
IppStatus ippQuantIntraInit_MPEG4(const Ipp8u* pQuantMatrix,  
    IppiQuantIntraSpec_MPEG4* pSpec, int bitsPerPixel);  
IppStatus ippQuantInterInit_MPEG4(const Ipp8u* pQuantMatrix,  
    IppiQuantInterSpec_MPEG4* pSpec, int bitsPerPixel);
```

#### Parameters

<i>pQuantMatrix</i>	Pointer to quantization matrix size of 64.
<i>pSpec</i>	Pointer to the initialized specification structure IppiQuantIntraSpec_MPEG4 or IppiQuantInterSpec_MPEG4.
<i>bitsPerPixel</i>	Video data precision used for saturation of the result. Valid within the range [4, 12].

#### Description

The functions `ippQuantIntraInit_MPEG4` and `ippQuantInterInit_MPEG4` are declared in the `ippvc.h` file. These functions initialize specification structure `IppiQuantIntraSpec_MPEG4` or `IppiQuantInterSpec_MPEG4`. If *pQuantMatrix* is NULL, the second quantization method is used; otherwise, the first method is used.

These functions are used in the MPEG-4 encoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

#### Return Values

`ippStsNoErr` Indicates no error.

<code>ippStsNullPtrErr</code>	Indicates an error pointer <i>pSpec</i> is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition when <i>bitsPerPixel</i> is out of the range [4, 12].

---

## QuantIntraGetSize\_MPEG4, QuantInterGetSize\_MPEG4

*Return size of specification structures.*

---

### Syntax

```
IppStatus ippiQuantIntraGetSize_MPEG4(int* pSpecSize);
IppStatus ippiQuantInterGetSize_MPEG4(int* pSpecSize);
```

### Parameters

<i>pSpecSize</i>	Pointer to the resulting size of the initialized specification structure IppiQuantIntraSpec_MPEG4 or IppiQuantInterSpec_MPEG4.
------------------	---

### Description

The functions `ippiQuantIntraGetSize_MPEG4` and `ippiQuantInterGetSize_MPEG4` are declared in the `ippvc.h` file. These functions return a size of specification structure `IppiQuantIntraSpec_MPEG4` or `IppiQuantInterSpec_MPEG4`.

These functions are used in the MPEG-4 encoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error pointer <i>pSpecSize</i> is NULL.



## QuantIntra\_MPEG4, QuantInter\_MPEG4

*Perform inverse quantization on intra/inter coded block.*

### Syntax

```
IppStatus ippiQuantIntra_MPEG4_16s_C1I(Ipp16s* pCoeffs, const
    IppiQuantIntraSpec_MPEG4* pSpec, int QP, int* pCountNonZero, int blockType);
IppStatus ippiQuantIntra_MPEG4_16s_I(Ipp16s * pSrcDst, Ipp8u QP, int
    blockIndex, const int * pQPMatrix);
IppStatus ippiQuantInter_MPEG4_16s_C1I(Ipp16s* pCoeffs, const
    IppiQuantInterSpec_MPEG4* pSpec, int QP, int* pCountNonZero);
IppStatus ippiQuantInter_MPEG4_16s_I(Ipp16s * pSrcDst, Ipp8u QP, const int *
    pQPMatrix);
```

### Parameters

<i>pCoeffs</i>	Pointer to the quantized DCT coefficients.
<i>pSpec</i>	Pointer to the initialized specification structure IppiQuantIntraSpec_MPEG4 or IppiQuantInterSpec_MPEG4 initialized by ippiQuantIntraInit_MPEG4 or ippiQuantInterInit_MPEG4 respectively.
<i>QP</i>	Quantization parameter.
<i>pCountNonZero</i>	Pointer to the count of non-zero coefficients.
<i>blockType</i>	Indicates the block type. Takes one of the following values: IPPVC_BLOCK_LUMA - for luma and alpha blocks, IPPVC_BLOCK_CHROMA - for chroma blocks.
<i>pSrcDst</i>	Pointer to the input intra block coefficients; as an output argument points to the quantized intra block coefficients.
<i>blockIndex</i>	Block index indicating the component type and position as defined in subclause 6.1.3.8 of <a href="#">[ISO14496A]</a> . Furthermore, indexes 6 to 9 indicate the alpha blocks spatially corresponding to luminance blocks 0 to 3 in the same macroblock.

*pQPMatrix*                      The pointer, which is NULL, if the second inverse quantization method is used. If the first inverse quantization method is used, it points to the quantization weighting coefficient's buffer (for inter MB) whose first 64 elements are the quantization weighting matrix in Q0. The second 64 elements are their reciprocals in Q21.

## Description

The functions `ippiQuantIntra_MPEG4_16s_C1I` and `ippiQuantInter_MPEG4_16s_C1I` are declared in the `ippvc.h` file. The functions `ippiQuantIntra_MPEG4_16s_I` and `ippiQuantInter_MPEG4_16s_I` are declared in the `ippalign.h` file. All these functions perform quantization on intra/inter coded block.

For `ippiQuantIntra_MPEG4_16s_C1I` and `ippiQuantInter_MPEG4_16s_C1I` output coefficients are saturated to lie in range  $[-2047; 2047]$ . Also these functions calculate the number of non-zero coefficients after quantization.

These functions are used in the MPEG-4 encoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

For `ippiQuantIntra_MPEG4_16s_I` and `ippiQuantInter_MPEG4_16s_I` output coefficients are saturated to lie in range  $[-127, 127]$ , that is, these functions support only *bitsPerPixel*=8.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one pointer is NULL.
<code>ippStsQPErr</code>	Indicates an error condition if <i>QP</i> is less or equal to 0.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

---

## VLC Encoding

---

### EncodeDCIntra\_MPEG4

*Encodes one DC coefficient for intra coded block.*

---

#### Syntax

```
IppStatus ippiEncodeDCIntra_MPEG4_16s1u(Ipp16s DC, Ipp8u **ppBitStream, int
    *pBitOffset, int blockType);
```

#### Parameters

<i>DC</i>	DC coefficient to be encoded.
<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . The pointer is updated by the function.
<i>blockType</i>	Indicates the block type, takes one of the following values: IPPVC_BLOCK_LUMA - for luma and alpha blocks, IPPVC_BLOCK_CHROMA - for chroma blocks.

#### Description

The function `ippiEncodeDCIntra_MPEG4_16s1u` is declared in the `ippvc.h` header file. This function performs VLC encoding of the DC coefficient only for one intra coded block using tables B.13, B.14, B.15 specified in [\[ISO14496\]](#).

This function is used in the MPEG-4 encoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].

---

## EncodeCoeffsIntra\_MPEG4

*Encodes DCT coefficients for intra coded block.*

---

### Syntax

```
IppStatus ippiEncodeCoeffsIntra_MPEG4_16s1u(const Ipp16s *pCoeffs, Ipp8u  
**ppBitStream, int *pBitOffset, int countNonZero, int rvlcFlag, int  
noDCFlag, int scan);
```

### Parameters

<i>pCoeffs</i>	Pointer to the input coefficients.
<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>ppBitStream</i> . The pointer is updated by the function.
<i>countNonZero</i>	Number of non-zero coefficients.
<i>rvlcFlag</i>	Flag, when set to '0' indicates that VLC tables B.16, B.18, B.19, and B.21 <a href="#">[ISO14496]</a> are used in encoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24, and B.25 <a href="#">[ISO14496]</a> are used.
<i>noDCFlag</i>	Flag, when set to '0' indicates that <i>pCoeffs</i> is set starting with zero element, otherwise - with the first element.
<i>scan</i>	Type of the scan, takes one of the following values:  IPPVC_SCAN_NONE, indicating that no scan is performed, IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan. IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan,  See the corresponding enumerator on <a href="#">p.16-3</a> .

### Description

The function `ippiEncodeCoeffsIntra_MPEG4_16s1u` is declared in the `ippvc.h` header file. This function performs VLC encoding of the DC coefficient (if *noDCFlag* is 0) and AC coefficients for one intra coded block in scan order defined by *scan*.

This function is used in the MPEG-4 encoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].

---

## EncodeCoeffsInter\_MPEG4

*Encodes DCT coefficients for inter coded block.*

---

### Syntax

```
IppStatus ippiEncodeCoeffsInter_MPEG4_16s1u(const Ipp16s *pCoeffs, Ipp8u
**ppBitStream, int *pBitOffset, int *countNonZero, int rvlc, int scan);
```

### Parameters

<i>pCoeffs</i>	Pointer to the input coefficients.
<i>ppBitStream</i>	Pointer to the pointer to the current byte in the bitstream buffer. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . The pointer is updated by the function.
<i>countNonZero</i>	Number of non-zero coefficients.
<i>rvlc</i>	Flag, when set to '0' indicates that VLC tables B.17, B.18, B.20, and B.22 <a href="#">[ISO14496]</a> are used in encoding DCT coefficients, otherwise the reversible variable length tables B.23, B.24, and B.25 <a href="#">[ISO14496]</a> are used.
<i>scan</i>	Type of the scan, takes one of the following values: IPPVC_SCAN_NONE, indicating that no scan is performed, IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan. IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan, See the corresponding enumerator on <a href="#">p.16-3</a> .

## Description

The function `ippiEncodeCoefInter_MPEG4_16s1u` is declared in the `ippvc.h` header file. This function performs VLC encoding of the DCT coefficients for one inter coded block in scan order defined by *scan*.

This function is used in the MPEG-4 encoder included into IPP Samples. See [introduction](#) to MPEG-4 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].

---

## EncodeVLCZigzag\_IntraDCVLC\_MPEG4, EncodeVLCZigzag\_IntraACVLC\_MPEG4

*Perform zigzag scanning and VLC encoding for one intra block.*

---

## Syntax

```
IppStatus ippiEncodeVLCZigzag_IntraDCVLC_MPEG4_16s1u(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s *pQDctBlkCoef, Ipp8u predDir, Ipp8u pattern,
    IppVideoComponent videoComp);
IppStatus ippiEncodeVLCZigzag_IntraACVLC_MPEG4_16s1u(Ipp8u** ppBitStream, int*
    pBitOffset, Ipp16s *pQDctBlkCoef, Ipp8u predDir, Ipp8u pattern);
```

## Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream. <i>*ppBitStream</i> is updated after the block is encoded.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated after the block is encoded.
<i>pQDctBlkCoef</i>	Pointer to the quantized DCT coefficient.

---

<i>predDir</i>	AC prediction direction, which is used to decide the zigzag scan pattern. This takes one of the following values: IPP_VIDEO_NONE – AC prediction is not used. Performs classical zigzag scan. IPP_VIDEO_HORIZONTAL – Horizontal prediction. Performs alternate-vertical zigzag scan. IPP_VIDEO_VERTICAL – Vertical prediction. Performs alternate-horizontal zigzag scan.
<i>pattern</i>	Block pattern, which is used to decide whether this block is encoded.
<i>videoComp</i>	Video component type (luminance, chrominance) of the current block.

### Description

The functions `ippiEncodeVLCZigzag_IntraDCVLC_MPEG4_16s1u` and `ippiEncodeVLCZigzag_IntraACVLC_MPEG4_16s1u` are declared in the `ippalign.h` file. These functions perform zigzag scanning and VLC encoding for one intra block.

The `IntraDCVLC` version uses Intra DC VLC to encode Intra DC coefficients. The `IntraACVLC` version uses Intra AC VLC to encode Intra DC coefficients.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

---

## EncodeVLCZigzag\_Inter\_MPEG4

*Performs classical zigzag scanning and VLC encoding for one inter block.*

---

### Syntax

```
IppStatus ippiEncodeVLCZigzag_Inter_MPEG4_16s1u(Ipp8u **ppBitStream, int*  
    pBitOffset, Ipp16s *pQDctBlkCoef, Ipp8u pattern);
```

### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream. <i>*ppBitStream</i> is updated after the block is encoded.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated after the block is encoded.
<i>pQDctBlkCoef</i>	Pointer to the quantized DCT coefficient.
<i>pattern</i>	Block pattern, which is used to decide whether this block is encoded.

### Description

The function `ippiEncodeVLCZigzag_Inter_MPEG4_16s1u` is declared in the `ippalign.h` file. This function performs classical zigzag scanning and VLC encoding for one inter block.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

### Block Encoding

---

## TransRecBlockCoef\_inter\_MPEG4

*Implements DCT, quantizes DCT coefficients of the inter block, and reconstructs the texture residual in the process.*

---

### Syntax

```
IppStatus ippiTransRecBlockCoef_inter_MPEG4 (Ipp16s *pSrc, Ipp16s * pDst,
        Ipp16s * pRec, Ipp8u QP, const int * pQPMatrix);
```

### Parameters

<i>pSrc</i>	Pointer to the residuals to be encoded for the current InterBlock.
<i>pDst</i>	Pointer to the quantized DCT coefficients buffer.



<i>pRec</i>	Pointer to the reconstructed texture residuals.
<i>QP</i>	Quantization parameter.
<i>pQPMatrix</i>	The pointer, which is NULL, if the second inverse quantization method is used. If the first inverse quantization method is used, it points to the quantization weighting coefficient's buffer (for inter MB) whose first 64 elements are the quantization weighting matrix in Q0. The second 64 elements are their reciprocals in Q20.

### Description

The function `ippiTransRecBlockCoef_inter_MPEG4` is declared in the `ippalign.h` file. This function implements DCT, quantizes DCT coefficients of the inter block, and reconstructs the texture residual in the process. There is no boundary check for the bitstream buffer.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

---

## TransRecBlockCoef\_intra\_MPEG4

*Quantizes DCT coefficients, implements AC/DC coefficients prediction of the intra block, and stores them into buffer.*

---

### Syntax

```
IppStatus ippiTransRecBlockCoef_intra_MPEG4 (Ipp8u *pSrc, Ipp16s * pDst, Ipp8u
    * pRec, Ipp16s *pPredBufRow, Ipp16s *pPredBufCol, Ipp16s * pPreACPredict,
    int *pSumErr, int blockIndex, Ipp8u QP, Ipp8u *pQpBuf, int srcStep, int
    dstStep, const int * pQPMatrix);
```

### Parameters

<i>pSrc</i>	Pointer to the pixels of current IntraBlock.
<i>pDst</i>	Pointer to the quantized DCT coefficients buffer.
<i>pRec</i>	Pointer to the reconstructed texture.

<i>pPredBufRow</i>	Pointer to the coefficient row buffer. As an output parameter points to the updated coefficient row buffer.
<i>pPredBufCol</i>	Pointer to the coefficient column buffer. As an output parameter points to the updated coefficient column buffer.
<i>pPreACPredict</i>	Pointer to the predicted coefficients buffer. The first data indicates the predicted direction of the current block.
<i>pSumErr</i>	Pointer to the sum of difference between predicted and unpredicted coefficients. As an output parameter points to the updated sum of the difference between predicted and unpredicted coefficients.
<i>blockIndex</i>	Block index indicating the component type and position as defined in subclause 6.1.3.8, of <a href="#">[ISO14496A]</a> . Furthermore, indexes 6 to 9 indicate the alpha blocks spatially corresponding to luminance blocks 0 to 3 in the same macroblock.
<i>QP</i>	Quantization parameter of the macroblock, which the current block belongs to.
<i>pQpBuf</i>	Pointer to the quantization parameter buffer.
<i>srcStep</i>	Width of the source buffer.
<i>dstStep</i>	Width of the reconstructed destination buffer.
<i>pQPMatrix</i>	The pointer, which is NULL, if the second inverse quantization method is used. If the first inverse quantization method is used, it points to the quantization weighting coefficient's buffer (for intra MB) whose first 64 elements are the quantization weighting matrix in Q0. The second 64 elements are their reciprocals in Q20.

## Description

The function `ippiTransRecBlockCoef_intra_MPEG4` is declared in the `ippalign.h` file. This function quantizes DCT coefficients, implements AC/DC coefficients prediction of the intra block, and stores them into buffer. The texture data is reconstructed for the next frame prediction.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

## MV Encoding

### FindMVPred\_MPEG4

*Finds the vector predictor from three candidates.*

#### Syntax

```
IppStatus ippiFindMVPred_MPEG4 (IppMotionVector* pSrcMVCurMB, IppMotionVector*
    pSrcCandMV1, IppMotionVector* pSrcCandMV2, IppMotionVector* pSrcCandMV3,
    Ipp8u* pSrcCandTransp1, Ipp8u* pSrcCandTransp2, Ipp8u* pSrcCandTransp3,
    Ipp8u* pSrcTranspCurr, IppMotionVector* pDstMVPred, IppMotionVector*
    pDstMVPredME, int blockIndex);
```

#### Parameters

<i>pSrcMVCurMB</i>	Pointer to the current <i>Y</i> macroblock buffers.
<i>pSrcCandMV1</i>	Pointers to the left candidate motion vector buffers.
<i>pSrcCandMV2</i>	Pointers to the top candidate motion vector buffers.
<i>pSrcCandMV3</i>	Pointers to the right-top candidate motion vector buffers.
<i>pSrcCandTransp1</i>	Pointers to the transparent status buffers of the corresponding macroblock or block.
<i>pSrcCandTransp2</i>	Pointers to the transparent status buffers of the corresponding macroblock or block.
<i>pSrcCandTransp3</i>	Pointers to the transparent status buffers of the corresponding macroblock or block.
<i>pSrcTranspCurr</i>	Pointers to the transparent status buffers of the current macroblock or block.
<i>pDstMVPred</i>	Pointer to the predicted motion vector.
<i>pDstMVPredME</i>	Pointer to three motion vector candidates, used only to determine motion activity when MVFAST is selected.
<i>blockIndex</i>	Index of a block in the current macroblock.

## Description

The function `ippiFindMVPred_MPEG4` is declared in the `ippalign.h` file. This function finds the vector predictor from three candidates and outputs three candidates for MVFAST, if `pDstMVPredME` is not NULL.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

---

## EncodeMV\_MPEG4

*Finds the prediction MV and encodes the difference.*

---

## Syntax

```
IppStatus ippiEncodeMV_MPEG4_8u16s(Ipp8u **ppBitStream, int *pBitOffset,  
    IppMotionVector * pMVCurMB, IppMotionVector * pSrcMVLeftMB, IppMotionVector  
    * pSrcMVUpperMB, IppMotionVector * pSrcMVUpperRightMB, Ipp8u * pTranspCurMB,  
    Ipp8u * pTranspLeftMB, Ipp8u * pTranspUpperMB, Ipp8u* pTranspUpperRightMB,  
    int fCodeForward, IppMacroblockType MBType);
```

## Parameters

<code>ppBitStream</code>	Pointer to the pointer to the current byte in the bitstream buffer.
<code>pBitOffset</code>	Pointer to the bit position in the byte pointed by <code>*ppBitStream</code> . Valid within 0 to 7.
<code>pMVCurMB</code>	Pointer to the current macroblock motion vector.
<code>pSrcMVLeftMB</code>	Pointer to the source left macroblock motion vector.
<code>pSrcMVUpperMB</code>	Pointer to the source upper macroblock motion vector.
<code>pSrcMVUpperRightMB</code>	Pointer to the source upper right macroblock motion vector.
<code>pTranspCurMB</code>	Pointer to the transparent status buffers of the current macroblock.
<code>pTranspLeftMB</code>	Pointer to the transparent status buffers of the source left macroblock.
<code>pTranspUpperMB</code>	Pointer to the transparent status buffers of the source upper macroblock.

---

<i>pTranspUpperRightMB</i>	Pointer to the transparent status buffers of the source upper right macroblock.
<i>fCodeForward</i>	Integer with values from 1 to 7; used in encoding motion vectors related to the search range.
<i>MBType</i>	Macroblock type, taking values from 0 to 9.

## Description

The function `ippiEncodeMV_MPEG4_8u16s` is declared in the `ippalign.h` file. This function finds the prediction MV and encodes the difference.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsBadArgErr</code>	Indicates bad arguments.

## H.261

This section contains IppVC functions for encoding and decoding video data according to ITU-T Recommendation H.261 ([[ITUH261](#)]).

**Table 16-19 H.261 Functions**

Function Short Name	Description
<b>Video Decoding Functions</b>	
<a href="#">DecodeCoeffsIntra_H261</a>	Decodes DCT coefficients for intra coded block.
<a href="#">DecodeCoeffsInter_H261</a>	Decodes DCT coefficients for inter coded block.
<a href="#">ReconstructCoeffsIntra_H261</a>	Reconstructs DCT coefficients for intra coded block.
<a href="#">ReconstructCoeffsInter_H261</a>	Reconstructs DCT coefficients for inter coded block.
<b>Video Encoding Functions</b>	
<a href="#">EncodeCoeffsIntra_H261</a>	Encodes and puts quantized DCT coefficients for intra coded block into bitstream.
<a href="#">EncodeCoeffsInter_H261</a>	Encodes and puts quantized DCT coefficients for inter coded block into bitstream.
<a href="#">Filter8x8_H261</a>	Performs two-dimensional filtering on a block.

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from

<http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm> .

## H.261 Decoder Functions

This section describes IPP functions that support the decoder and common operations specific to ITU-T Recommendation H.261.

The implemented IPP functions cover the following aspects of H.261 Decoder:

- Block layer coefficient decoding, including bitstream parsing, VLC decoding, inverse quantization (combined with VLC decoding), inverse zigzag positioning (combined with VLC decoding), with appropriate clipping at each step
- Two-dimensional loop filtering.

The following subsections of this section give a high-level description of the H.261 Decoder functions hierarchy, followed by macro/data structures used in the functions and detailed descriptions of individual functions.

## Decoding INTRA and INTER Macroblocks

Figure 16-42 INTRA Macroblock Decoding

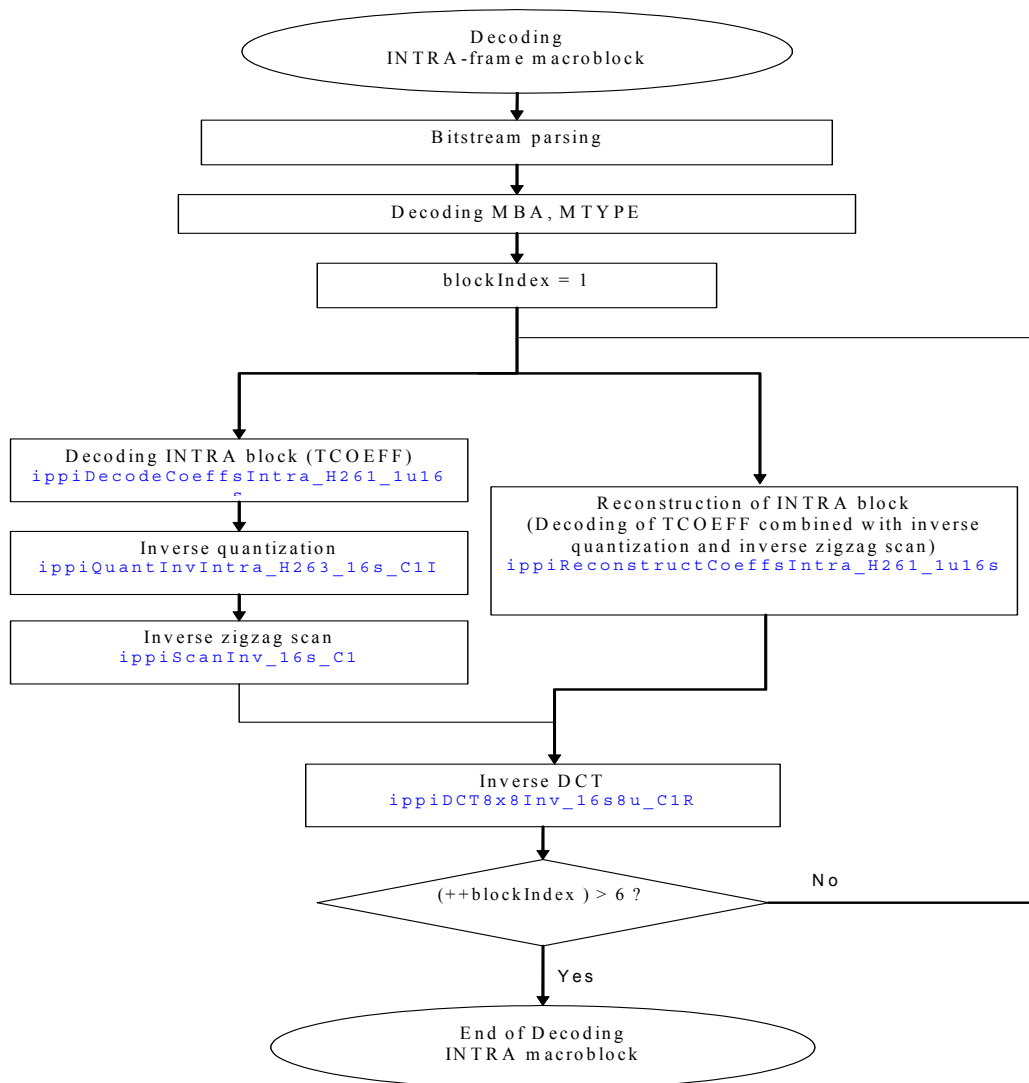
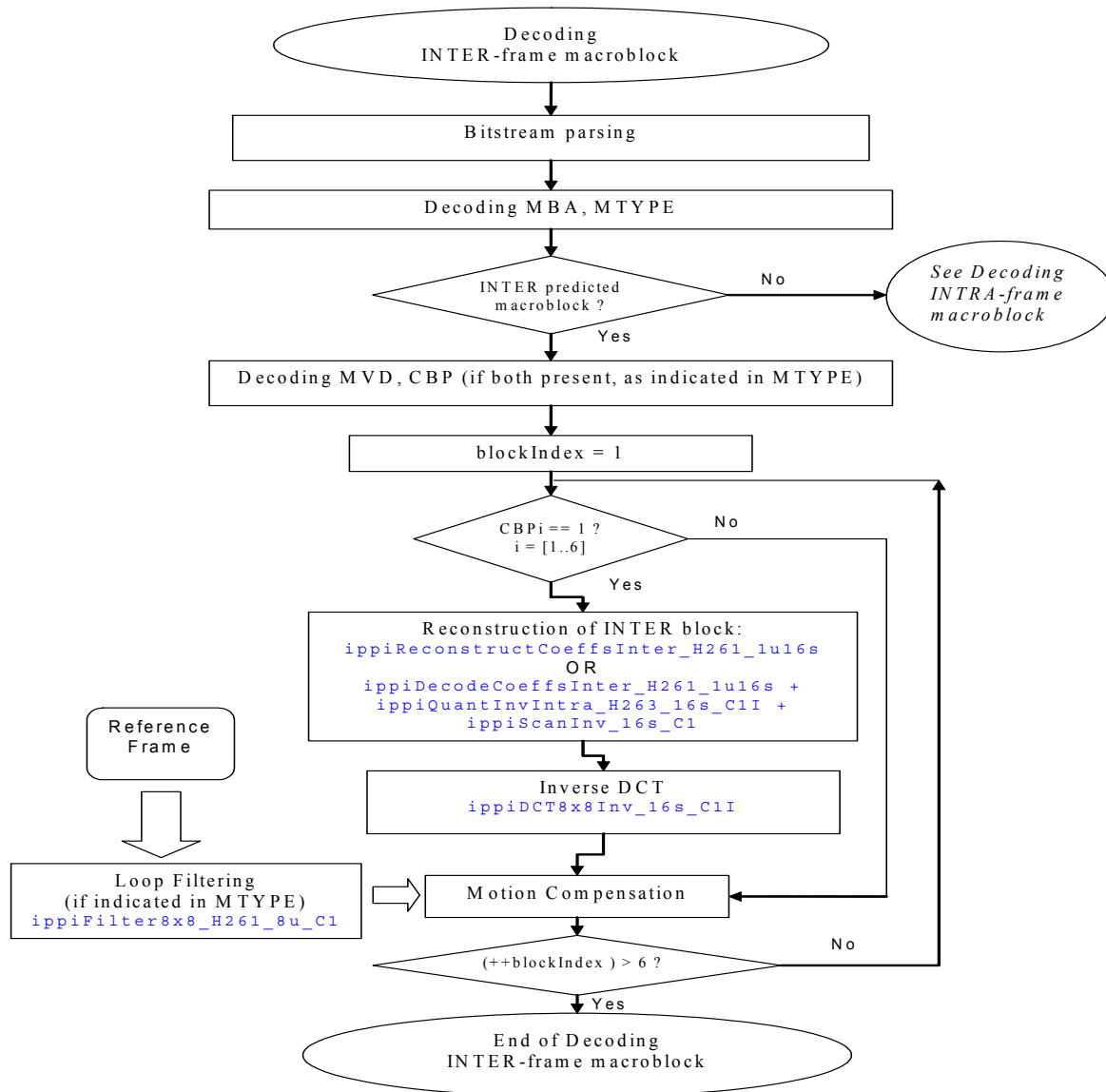




Figure 16-43 shows the process of INTER-frame macroblock decoding.

**Figure 16-43 INTER Macroblock Decoding**



## Structure and Macro Definitions

Analogous to [H.263](#) in the default prediction mode, horizontal and vertical components of a motion vector are integer values in the range [-15, 15]. Frames are never expanded in H.261, as motion vectors are always restricted to a point within the coded picture area.

---

## DecodeCoeffsIntra\_H261

*Decodes DCT coefficients for intra coded block.*

---

### Syntax

```
ippiDecodeCoeffsIntra_H261_1u16s(Ipp8u **ppBitStream, int *pBitOffset, Ipp16s  
    *pCoef, int *pIndxLastNonZero, int scan);
```

### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients. <i>pCoef</i> [0] is the DC coefficient.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>scan</i>	Type of the inverse scan, takes one of the following values:  IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_NONE, indicating that no inverse scan is to be performed.  See the corresponding enumerator on <a href="#">p.16-3</a> .

### Description

The function `ippiDecodeCoeffsIntra_H261_1u16s` is declared in the `ippvc.h` header file. This function performs decoding and, optionally, inverse scan of the DCT coefficients (DC and AC) for one Intra coded block. DC fixed length and AC VLC decoding processes are specified in

[ITUH261], subclause 4.2.4.1. If *scan* is not set to IPPVC\_SCAN\_NONE, the DCT coefficients, encoded in the bitstream in the classical zigzag scan order ([ITUH261], Figure12), are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the stream processing.

---

## DecodeCoeffsInter\_H261

*Decodes DCT coefficients for inter coded block.*

---

### Syntax

```
ippiDecodeCoeffsInter_H261_1u16s(Ipp8u **ppBitStream, int *pBitOffset, Ipp16s
    *pCoef, int *pIndxLastNonZero, int scan);
```

### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>scan</i>	Type of the inverse scan, takes one of the following values:

IPPVC\_SCAN\_ZIGZAG, indicating the classical zigzag scan,  
 IPPVC\_SCAN\_NONE, indicating that no inverse scan is to be performed.

See the corresponding enumerator on [p.16-3](#).

## Description

The function `ippiDecodeCoeffsInter_H261_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding and, optionally, inverse scan of the DCT coefficients for one Inter coded block as specified in [ITUH261], subclause 4.2.4.1 (Table 4). If `scan` is not set to `IPPVC_SCAN_NONE`, the DCT coefficients, encoded in the bitstream in the classical zigzag scan order ([ITUH261], Figure12), are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range [0, 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the VLC stream processing.

---

## ReconstructCoeffsIntra\_H261

*Reconstructs DCT coefficients for intra coded block.*

---

## Syntax

```
ippiReconstructCoeffsIntra_H261_1u16s(Ipp8u **ppBitStream, int *pBitOffset,
    Ipp16s *pCoef, int *pIndxLastNonZero, int QP);
```

## Parameters

<code>ppBitStream</code>	Pointer to pointer to the current byte in the bitstream buffer. <code>*ppBitStream</code> is updated by the function.
<code>pBitOffset</code>	Pointer to the bit position in the byte pointed by <code>*ppBitStream</code> . Valid within the range 0 to 7. <code>*pBitOffset</code> is updated by the function.

---

<i>pCoef</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>QP</i>	Quantization parameter.

## Description

The function `ippiReconstructCoeffsIntra_H261_1u16s` is declared in the `ippvc.h` header file. This function performs decoding, dequantization, and inverse scanning of the DCT coefficients for one Intra coded block.

DCT coefficients decoding and dequantization processes are specified in [ITUH261], subclause 4.2.4.1. The DCT coefficients, encoded in the bitstream in classical zigzag scan order ([ITUH261], Figure 12), are reordered in the function into the normal order, that is, the order, in which the coefficients are arranged on DCT output.

This function is used in the H.261 decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the stream processing.
<code>ippStsQPErr</code>	Indicates an error condition if <i>QP</i> is out of the range [1, 31].

---

## ReconstructCoeffsInter\_H261

*Reconstructs DCT coefficients for inter coded block.*

---

### Syntax

```
ippiReconstructCoeffsInter_H261_1u16s(Ipp8u **ppBitStream, int *pBitOffset,  
    Ipp16s *pCoef, int *pIndxLastNonZero, int QP);
```

### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>QP</i>	Quantization parameter.

### Description

The function `ippiReconstructCoeffsInter_H261_1u16s` is declared in the `ippvc.h` header file. This function performs decoding, dequantization and inverse scanning of the DCT coefficients for one Inter coded block.

DCT coefficients decoding and dequantization processes are specified in [ITUH261], subclause 4.2.4.1. The DCT coefficients, encoded in the bitstream in classical zigzag scan order ([ITUH261], Figure 12), are reordered in the function into the normal order, that is, the order, in which the coefficients are arranged on DCT output.

This function is used in the H.261 decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

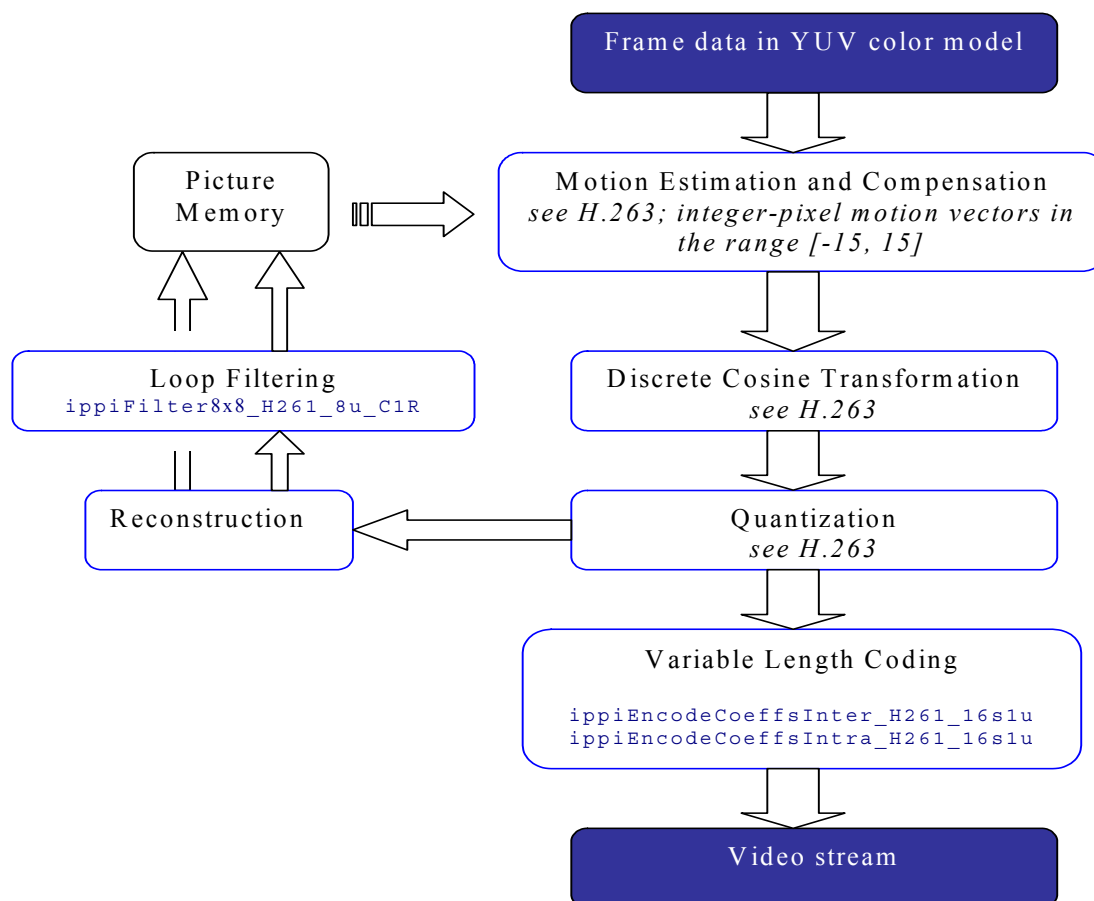
<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is <code>NULL</code> .
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range <code>[0, 7]</code> .
<code>ippStsVLCCodeErr</code>	Indicates an error condition if an illegal code is detected through the stream processing.
<code>ippStsQPErr</code>	Indicates an error condition if <code>QP</code> is out of the range <code>[1, 31]</code> .

## H.261 Encoder Functions

This section describes IPP functions that support the encoder operations specific to ITU-T Recommendation H.261. The implemented IPP functions cover the block layer coefficient encoding, including VLC encoding and bitstream forming. [Figure 16-44](#) shows H.261 video encoding pipeline.

**Figure 16-44** H.261 Encoding Pipeline.





---

## EncodeCoeffsIntra\_H261

*Encodes and puts quantized DCT coefficients for intra coded block into bitstream.*

---

### Syntax

```
IppStatus ippiEncodeCoeffsIntra_H261_16s1u(Ipp16s *pQCoef, Ipp8u **ppBitStream,
int *pBitOffset, int countNonZero, int scan);
```

### Parameters

<i>pQCoef</i>	Pointer to the array of quantized DCT coefficients. <i>pQCoef</i> [0] is the DC coefficient.
<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>countNonZero</i>	Number of non-zero coefficients in the block. Valid within the range 1 to 64.
<i>scan</i>	Type of the scan to be performed on the coefficients before encoding, takes one of the following values:  IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_NONE, indicating that no scan is to be performed, that is, the input coefficients are already in the scan order.  See the corresponding enumerator on <a href="#">p.16-3</a> .

### Description

The function `ippiEncodeCoeffsIntra_H261_16s1u` is declared in the `ippvc.h` header file. This function performs encoding of the quantized DCT coefficients in a scan order for one Intra coded block and puts the codes into the bitstream. DC fixed length and AC VLC encoding processes are specified in [ITUH261], subclause 4.2.4.1.

This function is used in the H.261 encoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is <code>NULL</code> .
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range <code>[0, 7]</code> .
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <code>countNonZero</code> is out of the range <code>[1, 64]</code> .

---

## EncodeCoeffsInter\_H261

*Encodes and puts quantized DCT coefficients for inter coded block into bitstream.*

---

## Syntax

```
IppStatus ippIEncodeCoeffsInter_H261_16s1u(Ipp16s *pQCoef, Ipp8u **ppBitStream,
      int *pBitOffset, int countNonZero, int scan);
```

## Parameters

<code>pQCoef</code>	Pointer to the array of quantized DCT coefficients. <code>pQCoef[0]</code> is the DC coefficient.
<code>ppBitStream</code>	Pointer to pointer to the current byte in the bitstream buffer. <code>*ppBitStream</code> is updated by the function.
<code>pBitOffset</code>	Pointer to the bit position in the byte pointed by <code>*ppBitStream</code> . Valid within the range 0 to 7. <code>*pBitOffset</code> is updated by the function.
<code>countNonZero</code>	Number of non-zero coefficients in the block. Valid within the range 1 to 64.
<code>scan</code>	Type of the scan to be performed on the coefficients before encoding, takes one of the following values:  <code>IPPVC_SCAN_ZIGZAG</code> , indicating the classical zigzag scan, <code>IPPVC_SCAN_NONE</code> , indicating that no scan is to be performed, that is, the input coefficients are already in the scan order.  See the corresponding enumerator on <a href="#">p.16-3</a> .

## Description

The function `ippiEncodeCoeffsInter_H261_16s1u` is declared in the `ippvc.h` header file. This function performs VLC encoding of the quantized DCT coefficients in a scan order for one Inter coded block and puts the codes into the bitstream. The encoding process is specified in [\[ITUH261\]](#), subclause 4.2.4.1.

This function is used in the H.261 encoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is <code>NULL</code> .
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range <code>[0, 7]</code> .
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <code>countNonZero</code> is out of the range <code>[1, 64]</code> .

---

## Filter8x8\_H261

*Performs two-dimensional filtering on a block.*

---

## Syntax

```
ippiFilter8x8_H261_8u_C1R(Ipp8u *pSrc, int srcStep, Ipp8u *pDst, int dstStep);
```

## Parameters

<code>pSrc</code>	Pointer to the origin of the source block.
<code>srcStep</code>	Width in bytes of the source image plane.
<code>pDst</code>	Pointer to the origin of the destination block.
<code>dstStep</code>	Width in bytes of the destination image plane.

### Description

The function `ippiFilter8x8_H261_8u_C1R` is declared in the `ippvc.h` header file. This function performs filtering on an 8x8 block as specified in [\[ITUH261\]](#), subclause 3.2.3. The two-dimensional filter may be used in both encoder and decoder to modify the prediction. The application of the filter to a macroblock (to all six blocks) is signaled in the MTYPE (macroblock type) codeword.

This function is used in the H.261 encoder and decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## H.263

This section contains ippVC functions for encoding and decoding of video data according to ITU-T Recommendation H.263 ([ITUH263](#)) and Annexes, which is most often denoted by the “H263 + decoder” acronym.

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm> .

Table 16-20 H.263 Video Decoder Functions

Function Short Name	Description
<b>VLC Decoding</b>	
<a href="#">DecodeBlockCoef_Intra_H263</a>	Decodes the INTRA block coefficients.
<a href="#">DecodeBlockCoef_Inter_H263</a>	Decodes the INTER block coefficients.
<a href="#">DecodeDCIntra_H263</a>	Decodes DC coefficient for intra coded block.
<a href="#">DecodeCoeffsIntra_H263</a>	Decodes AC coefficients for intra coded block.
<a href="#">DecodeCoeffsInter_H263</a>	Decodes DCT coefficients for inter coded block.
<b>Quantization</b>	
<a href="#">QuantInvIntra_H263</a>	Performs inverse quantization on an intra coded block stored in a one-dimensional buffer.
<a href="#">QuantInvInter_H263</a>	Performs inverse quantization on an inter coded block stored in a one-dimensional buffer.
<b>Prediction</b>	
<a href="#">AddBackPredPB_H263</a>	Performs bidirectional prediction for a B-block in a PB-frame.
<b>Frame Expansion</b>	
<a href="#">ExpandFrame_H263</a>	Expands the frame to the plane.
<b>Resampling</b>	
<a href="#">Resample_H263</a>	Performs picture resampling defined in terms of picture area corner displacements.
<a href="#">UpsampleFour_H263</a>	Performs factor-of-4 picture upsampling.
<a href="#">DownsampleFour_H263</a>	Performs factor-of-4 picture downsampling.
<a href="#">UpsampleFour8x8_H263</a>	Performs factor-of-4 upsampling on an 8x8 block.

**Table 16-20 H.263 Video Decoder Functions (continued)**

Function Short Name	Description
<a href="#"><u>SpatialInterpolation H263</u></a>	Interpolates a picture by a factor of two horizontally, vertically, or both horizontally and vertically.
<b>Boundary Filtering</b>	
<a href="#"><u>FilterBlockBoundaryHorEdge H263,</u></a> <a href="#"><u>FilterBlockBoundaryVerEdge H263</u></a>	Perform block boundary filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.
<a href="#"><u>FilterDeblocking8x8HorEdge H263,</u></a> <a href="#"><u>FilterDeblocking8x8VerEdge H263</u></a>	Perform deblocking filtering of one block edge on the reconstructed frames.
<a href="#"><u>FilterDeblocking16x16HorEdge H263,</u></a> <a href="#"><u>FilterDeblocking16x16VerEdge H263</u></a>	Perform deblocking filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.
<b>Middle Level Functions</b>	
<a href="#"><u>ReconstructCoeffsIntra H263</u></a>	Reconstructs DCT coefficients for an intra coded block.
<a href="#"><u>ReconstructCoeffsInter_H263</u></a>	Reconstructs DCT coefficients for an inter coded block.

**Table 16-21 H.263 Video Encoder Functions**

Function Short Name	Description
<b>VLC Encoding</b>	
<a href="#"><u>EncodeDCIntra H263</u></a>	Encodes and puts a quantized DC coefficient for an intra coded block into bitstream.
<a href="#"><u>EncodeCoeffsIntra_H263</u></a>	Encodes and puts quantized DCT coefficients for an intra coded block into bitstream.
<a href="#"><u>EncodeCoeffsInter H263</u></a>	Encodes and puts quantized DCT coefficients for inter coded block into bitstream.
<b>Quantization</b>	
<a href="#"><u>QuantIntra H263</u></a>	Performs quantization on an intra coded block.
<a href="#"><u>QuantInter H263</u></a>	Performs quantization on an inter coded block.

Table 16-21 H.263 Video Encoder Functions (continued)

Function Short Name	Description
<b>Resampling</b> <a href="#">DownsampleFour16x16_H263</a>	Performs factor-of-4 downsampling on a 16x16 block.

H.263 Decoder Functions

This section describes Intel IPP functions that support general video processing and the decoder part of ITU-T Recommendation H.263 (see ([ITUH263]) and Annexes, which is often denoted by the “H263+ decoder” acronym.

The implemented Intel IPP functions cover the following aspects of H.263+ Decoder:

- Inverse quantization, inverse zigzag positioning, reconstruction and IDCT
- Block layer coefficient decoding, including bitstream parsing, VLC decoding, inverse quantization, inverse zigzag positioning and IDCT, with appropriate clipping on each step
- Unrestricted Motion Vectors mode (Annex D/H.263+)
- Advanced Prediction mode (Annex F/H.263+)
- PB-frames mode (Annex G/H.263+)
- Advanced Intra-Coding mode (Annex I/H.263+)
- Deblocking Filter mode (Annex J/H.263+)
- Temporal, SNR, and Spatial Scalability mode (AnnexO/H.263+)
- Reference Picture Resampling (Annex P/H.263+)
- Reduced-Resolution Update mode (Annex Q/H.263+)
- Alternative INTER VLC mode (Annex S/H.263+)
- Modified Quantization mode (Annex T/H.263+).

The following subsections of this section give a high-level description of the H.263+ Decoder functions hierarchy, followed by macro/data structures used in the functions and the detailed descriptions of individual functions.

## INTRA and INTER Macroblocks Decoding

Figure 16-45 INTRA Macroblock Decoding

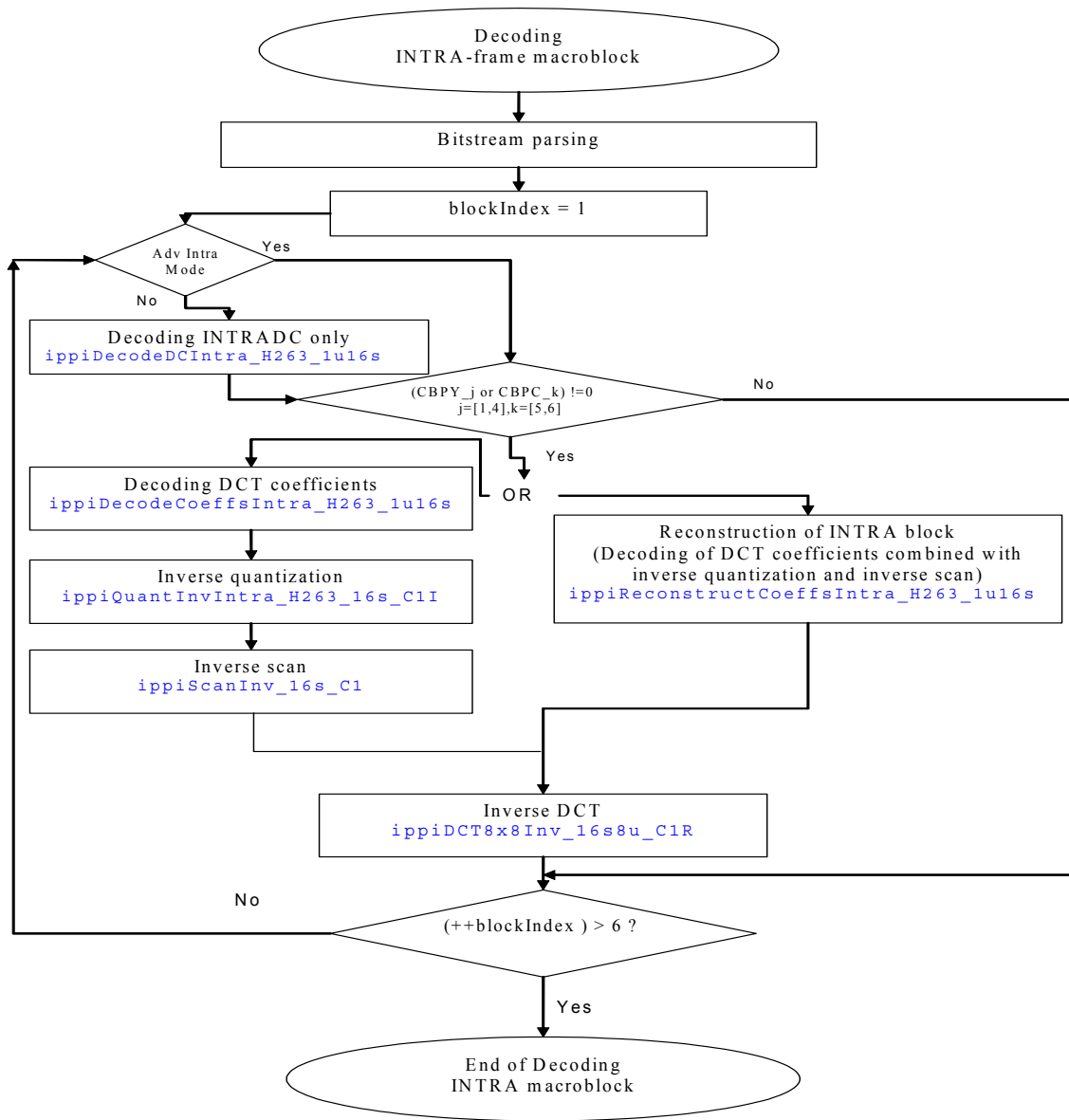
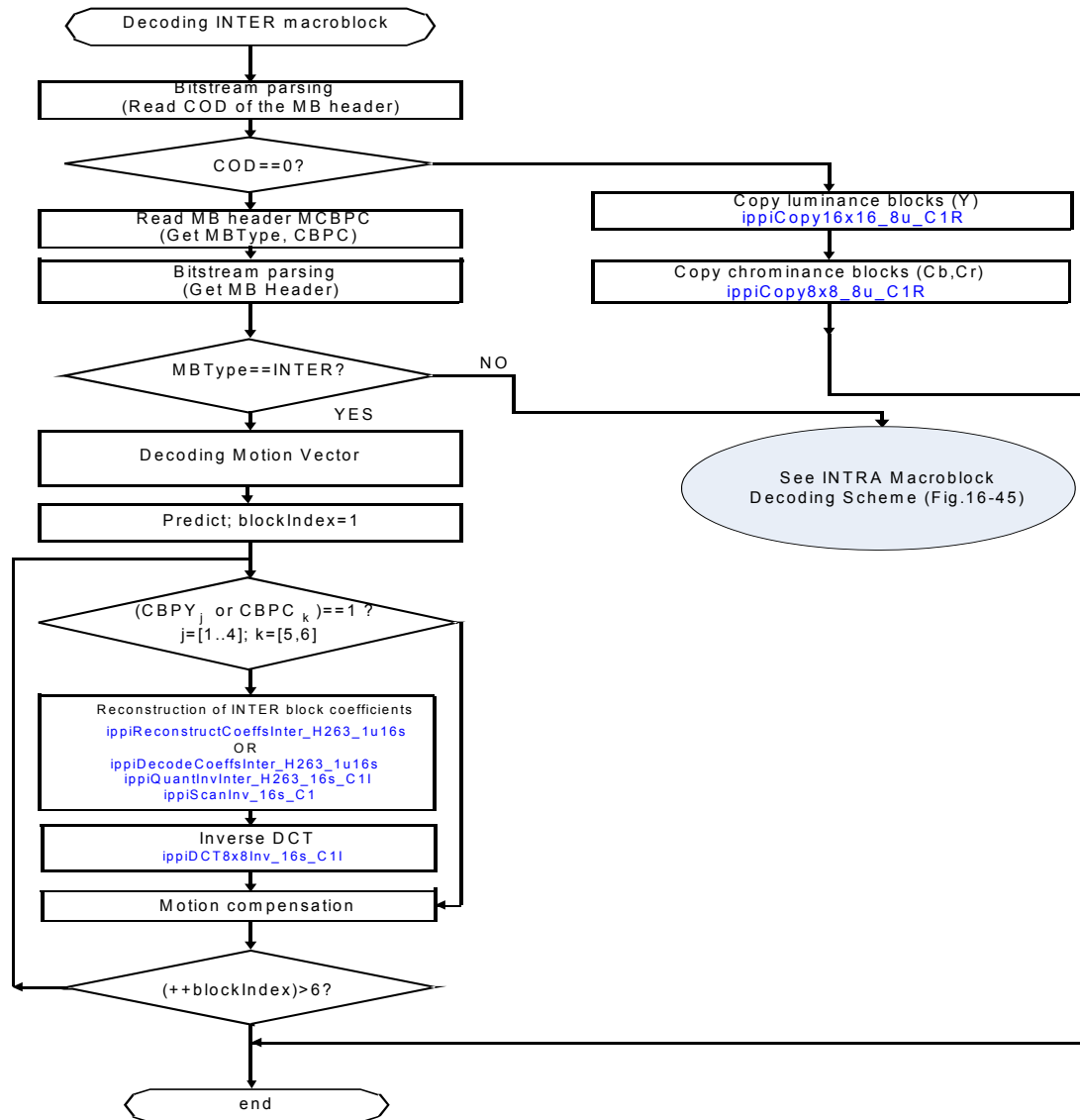




Figure 16-46 shows the process of INTER macroblock decoding.

**Figure 16-46 INTER Macroblock Decoding**



## VLC Decoding

---

### DecodeBlockCoef\_Intra\_H263

*Decodes the INTRA block coefficients.*

---

#### Syntax

```
IppStatus ippiDecodeBlockCoef_Intra_H263_1u8u(Ipp8u** ppBitStream,
        int pBitOffset, Ipp8u* pDst, int step, int QP);
```

#### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer ( <i>*ppBitStream</i> is updated after the block is decoded).
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated after the block is decoded.
<i>pDst</i>	Pointer to the block in the destination plane.
<i>step</i>	Width of the destination plane.
<i>QP</i>	Quantization parameter (for non-INTRADC coefficients).

#### Description

The function `ippiDecodeBlockCoef_Intra_H263_1u8u` is declared in the `ippalign.h` file. This function decodes the INTRA block coefficients. Inverse quantization, inverse zigzag positioning (classical) and IDCT, with appropriate clipping on each step, are performed on the coefficients. The results are then placed in the output frame/plane on a pixel basis.

Note that no boundary check for the bitstream buffer is done.

For INTRA block, the output values are clipped to [0, 255] and written to the current frame within the destination plane.

This function is used only when at least one non-zero AC coefficient of the current block exists in the bitstream.



**CAUTION.** Illegal code in bitstream which can not be looked up in VLC table could result in status error.

Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsH263BlockStepErr</code>	Indicates an error condition if the <i>step</i> value is less than 8.
<code>ippStsH263QuantErr</code>	Indicates an error condition if the quantizer value has a zero or negative value, or if it is greater than 31.

DecodeBlockCoef\_Inter\_H263

*Decodes the INTER block coefficients.*

Syntax

```
IppStatus ippiDecodeBlockCoef_Inter_H263_1u16s(Ipp8u** ppBitStream, int* pBitOffset, Ipp16s* pDst, int QP);
```

Parameters

<code>ppBitStream</code>	Pointer to pointer to the current byte in the bitstream buffer ( <i>*ppBitStream</i> is updated after the block is decoded).
<code>pBitOffset</code>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated after the block is decoded.
<code>pDst</code>	Pointer to the decoded residual buffer (a contiguous array of 64 short integers).
<code>QP</code>	Quantization parameter.

## Description

The function `ippiDecodeBlockCoef_Inter_H263_1u16s` is declared in the `ippalign.h` file. This function decodes the INTER block coefficients. Inverse quantization, inverse zigzag positioning (classical) and IDCT, with appropriate clipping on each step, are performed on the coefficients. The results (residuals) are placed in a contiguous array of 64 short integers.

Note that no boundary check for the bitstream buffer is done.

For INTER block, the output buffer holds the residuals for further reconstruction.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsH263QuantErr</code>	Indicates an error condition if the quantizer value has a zero or negative value, or if it is greater than 31.

---

## DecodeDCIntra\_H263

*Decodes DC coefficient for intra coded block.*

---

## Syntax

```
IppStatus ippiDecodeDCIntra_H263_1u16s(Ipp8u **ppBitStream, int *pBitOffset,
    Ipp16s *pDC);
```

## Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pDC</i>	Pointer to the output coefficient.

## Description

The function `ippiDecodeDCIntra_H263_1u16s` is declared in the `ippvc.h` header file. This function performs fixed length decoding of the DC coefficient for one Intra coded block. Intra DC decoding process is specified in [ITUH263], subclause 5.4.1.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCErr</code>	Indicates an error condition if an illegal code is detected through the DC stream processing.

---

## DecodeCoeffsIntra\_H263

*Decodes AC coefficients for intra coded block.*

---

## Syntax

```
IppStatus ippiDecodeCoeffsIntra_H263_1u16s(Ipp8u **ppBitStream, int
    *pBitOffset, Ipp16s *pCoef, int *pIndxLastNonZero, int advIntraFlag, int
    modQuantFlag, int scan);
```

## Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients. <i>pCoef</i> [0] is the DC coefficient.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1 when in Advanced Intra Coding mode, and to 0 otherwise.

<i>advIntraFlag</i>	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.
<i>scan</i>	Type of the inverse scan, takes one of the following values:  IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan, IPPVC_SCAN_NONE, indicating that no inverse scan is to be performed.  See the corresponding enumerator on <a href="#">p.16-3</a> .

### Description

The function `ippiDecodeCoeffsIntra_H263_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding and, optionally, inverse scan of the AC coefficients for one Intra coded block. Intra AC VLC decoding process is specified in [ITUH263], subclause 5.4.2, and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. When in Advanced Intra Coding mode, VLC Table I.2 from [ITUH263] Annex I is used for all Intra DC and Intra AC coefficients, otherwise Table 16 [ITUH263] is used to decode AC coefficients (starting from `pCoef[1]`) only. If *scan* is not set to `IPPVC_SCAN_NONE`, the DCT coefficients, encoded in the bitstream in the classical zigzag, alternate-horizontal, or alternate-vertical scan order, are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output. The three scan patterns are shown in [ITUH263], Figure14 and [ITUH263], Annex I, Figure I.2.

This function is used in the H.263 decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is <code>NULL</code> .
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range [0, 7].
<code>ippStsVLCErr</code>	Indicates an error condition if an illegal code is detected through the VLC stream processing.

## DecodeCoeffsInter\_H263

*Decodes DCT coefficients for inter coded block.*

### Syntax

```
IppStatus ippiDecodeCoeffsInter_H263_1u16s(Ipp8u **ppBitStream, int
    *pBitOffset, Ipp16s *pCoef, int *pIndxLastNonZero, int modQuantFlag, int
    scan);
```

### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.
<i>scan</i>	Type of the inverse scan, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_NONE, indicating that no inverse scan is to be performed. See the corresponding enumerator on <a href="#">p.16-3</a> .

### Description

The function `ippiDecodeCoeffsInter_H263_1u16s` is declared in the `ippvc.h` header file. This function performs VLC decoding and, optionally, inverse scan of the DCT coefficients for one Inter coded block. Inter DCT VLC decoding process is specified in [ITUH263], subclause 5.4.2 (Table 16), and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. If *scan* is not set to `IPPVC_SCAN_NONE`, the DCT coefficients,

encoded in the bitstream in the classical zigzag scan order, are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output. The zigzag scan pattern is shown in [ITUH263], Figure14.

This function is used in the H.263 decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCErr</code>	Indicates an error condition if an illegal code is detected through the VLC stream processing.

### Inverse Quantization

---

## QuantInvIntra\_H263

*Performs inverse quantization on an intra coded block stored in a one-dimensional buffer.*

---

### Syntax

```
IppStatus ippQuantInvIntra_H263_16s_C1I(Ipp16s *pSrcDst, int indxLastNonZero,
    int QP, int advIntraFlag, int modQuantFlag);
```

### Parameters

<i>pSrcDst</i>	Pointer to the coefficient buffer of the block. The parameter contains quantized coefficients on input and dequantized coefficients on output.
<i>indxLastNonZero</i>	Index of the last non-zero coefficient. This parameter provides faster operation. If the value is unknown, set to 63.
<i>QP</i>	Quantization parameter.



<i>advIntraFlag</i>	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

## Description

The function `ippiQuantInvIntra_H263_16s_C1I` is declared in the `ippvc.h` header file. This function performs inverse quantization on Intra coded block. When not in Advanced Intra Coding mode, the dequantization processes for the Intra DC and for all other non-zero coefficients are specified in [ITUH263], subclause 6.2.1, otherwise all the coefficients are dequantized as specified in [ITUH263] Annex I, clause I.3. When not in Advanced Intra Coding mode and not in Modified Quantization mode, the output coefficients other than the Intra DC are clipped to the range [-2048, 2047] ([ITUH263], subclause 6.2.2). The overall procedure is defined in [ITUH263] as:

```
if (advIntraFlag == 0):
```

$$pDst[i] = \begin{cases} 0, & pSrc[i] = 0 \\ \text{sign}(pSrc[i]) * QP * (2 * |pSrc[i]| + 1), & pSrc[i] \neq 0, \quad QP \text{ is odd} \\ \text{sign}(pSrc[i]) * (QP * (2 * |pSrc[i]| + 1) - 1), & pSrc[i] \neq 0, \quad QP \text{ is even} \end{cases}$$

where  $i = 1, 2, \dots, \text{indxLastNonZero}$

```
pDst[0] = 8 * pSrc[i]
```

```
if (advIntraFlag != 0):
```

```
pDst[i] = 2 * QP * pSrc[i]
```

where  $i = 0, 1, 2, \dots, \text{indxLastNonZero}$

```
if (advIntraFlag == 0 && modQuantFlag == 0):
```

```
pDst[i] = MIN(MAX(pDst[i], -2048), 2047).
```




---

**NOTE.** The function `ippiQuantInvIntra_H263_16s_C1I` can be applied to a buffer of arbitrary size (`indxLastNonZero` can be any positive number), and can thus be used, for example, to process multiple blocks in one call. In this case, for any Intra block following the first one, the Intra DC should be processed separately, if not in Advanced Intra mode.

---

This function is used in the H.261, H.263, and MPEG-4 encoders and decoders included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrcDst</code> is <code>NULL</code> .
<code>ippStsQPErr</code>	Indicates an error condition if <code>QP</code> is out of the range <code>[1, 31]</code> .
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <code>indxLastNonZero</code> is negative.

---

## QuantInvInter\_H263

*Performs inverse quantization on an inter coded block stored in a one-dimensional buffer.*

---

### Syntax

```
IppStatus ippiQuantInvInter_H263_16s_C1I(Ipp16s *pSrcDst, int indxLastNonZero,
int QP, int modQuantFlag);
```

### Parameters

<code>pSrcDst</code>	Pointer to the coefficient buffer of the block. The parameter contains quantized coefficients on input and dequantized coefficients on output.
<code>indxLastNonZero</code>	Index of the last non-zero coefficient. This parameter provides faster operation. If the value is unknown, set to 63.

<i>QP</i>	Quantization parameter.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

## Description

The function `ippiQuantInvInter_H263_16s_C1I` is declared in the `ippvc.h` header file. This function performs inverse quantization on Inter coded block. The dequantization process is specified in [ITUH263], subclause 6.2.1. When not in Modified Quantization mode, the output coefficients are clipped to the range [-2048, 2047] ([ITUH263], subclause 6.2.2). The overall procedure is defined in [ITUH263] as:

$$pDst[i] = \begin{cases} 0, & pSrc[i] = 0 \\ \text{Sign}(pSrc[i]) * QP * (2 * |pSrc[i]| + 1), & pSrc[i] \neq 0, \quad QP \text{ is odd} \\ \text{Sign}(pSrc[i]) * (QP * (2 * |pSrc[i]| + 1) - 1), & pSrc[i] \neq 0, \quad QP \text{ is even} \end{cases}$$

where  $i = 0, 1, 2, \dots, \text{indxLastNonZero}$

if (`modQuantFlag == 0`):

`pDst[i] = MIN(MAX(pDst[i], -2048), 2047).`



**NOTE.** The function `ippiQuantInvInter_H263_16s_C1I` can be applied to a buffer of arbitrary size (`indxLastNonZero` can be any positive number), and can thus be used, for example, to process multiple blocks in one call. In this case, for any Intra block following the first one, the Intra DC should be processed separately, if not in Advanced Intra mode.

This function is used in the H.261, H.263, and MPEG-4 encoders and decoders included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <code>pSrcDst</code> is NULL.

<code>ippStsQPErr</code>	Indicates an error condition if <i>QP</i> is out of the range [1, 31].
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <i>indxLastNonZero</i> is negative.

## Prediction

---

### AddBackPredPB\_H263

*Performs bidirectional prediction for a B-block in a PB-frame.*

---

#### Syntax

```
IppStatus ippAddBackPredPB_H263_8u_C1R(const Ipp8u *pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u *pSrcDst, int srcDstStep, int acc);
```

#### Parameters

<i>pSrc</i>	Pointer to the origin of the source image (P-macroblock) region of interest (ROI).
<i>srcStep</i>	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source image.
<i>srcRoiSize</i>	Size of the source ROI.
<i>pSrcDst</i>	Pointer to the origin of the source-destination image ROI, that is, bidirectionally-predicted part of the block.
<i>srcDstStep</i>	Width in bytes of the source-destination image plane.
<i>acc</i>	Parameter that defines pixel accuracy for backward prediction: bit 0 (the least significant bit) contains the half-pixel offset in horizontal direction, bit 1 – the offset in vertical direction.

#### Description

The function `ippAddBackPredPB_H263_8u_C1R` is declared in the `ippvc.h` header file. This function calculates backward prediction for a B-block of a PB-frame and adds it to the block, previously reconstructed with forward prediction. All the operations are restricted to the bidirectionally-predicted part of the B-block. The parameter *srcRoiSize* defines the area size.

The backward prediction is performed with pixel accuracy defined by *acc*, and the sum of the forward and backward predictions for every pixel within *srcRoiSize* is divided by 2 (division by truncation). The bidirectional prediction procedure is specified in [ITUH263], Annex G, clause G.5, bidirectionally- and forward-predicted areas for a B-block are demonstrated in [ITUH263], Annex G, Figure G.2.

This function is used in the H.263 decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> has a field with zero or negative value.

### Frame Expansion

---

## ExpandFrame\_H263

*Expands the frame to the plane.*

---

### Syntax

```
IppStatus ippIExpandFrame_H263_8u(Ipp8u* pSrcDstPlane, int frameWidth, int  
    frameHeight, int expandPels, int step);
```

### Parameters

<i>pSrcDstPlane</i>	Pointer to the plane.
<i>frameWidth</i>	Width of the frame.
<i>frameHeight</i>	Height of the frame.
<i>expandPels</i>	Number of pixels to be expanded in one direction.
<i>step</i>	Step value, $step \geq \text{planeWidth} = \text{frameWidth} + 2 * \text{expandPels}$ .

### Description

The function `ippiExpandFrame_H263_8u` is declared in the `ippalign.h` file. This function expands the frame to the plane in order to enable the motion vectors over picture boundaries feature. This feature should be enabled if the Annexes D, F, J of H263+ are supported. When a pixel referenced by a motion vector is outside the coded picture area, an edge pixel is used instead. It is assumed that the picture frame has been reconstructed prior to the function call.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrcDstPlane</i> pointer is NULL.
<code>ippStsH263FrameWidthErr</code>	Indicates an error condition if the <i>frameWidth</i> value is less than 8.
<code>ippStsH263FrameHeightErr</code>	Indicates an error condition if the <i>frameWidth</i> has a zero or negative value.
<code>ippStsH263ExpandPelsErr</code>	Indicates an error condition if the <i>expandPels</i> value is less than 8.
<code>ippStsH263BlockStepErr</code>	Indicates an error condition if the <i>step</i> value is less than 8.
<code>ippStsH263PlaneStepErr</code>	Indicates an error condition if the <i>step</i> value is less than the plane width.

### Resampling

---

## Resample\_H263

*Performs picture resampling defined in terms of picture area corner displacements.*

---

### Syntax

```
IppStatus ippiResample_H263_8u_P3R(const Ipp8u *pSrcY, int srcYStep, IppiSize  
ySrcRoiSize, const Ipp8u *pSrcCb, int srcCbStep, const Ipp8u *pSrcCr, int  
srcCrStep, Ipp8u *pDstY, int dstYStep, IppiSize dstYRoiSize, Ipp8u *pDstCb,  
int dstCbStep, Ipp8u *pDstCr, int dstCrStep, IppMotionVector warpParams[4],  
int wda, int rounding, int fillMode, int fillColor[3]);
```

## Parameters

<i>pSrcY</i>	Pointer to the origin of the source image region of interest (ROI) in the luminance plane.
<i>srcYStep</i>	Width in bytes of the source image luminance (Y) plane, that is, distance in bytes between the starting ends of consecutive lines of the source image in the luminance plane.
<i>ySrcRoiSize</i>	Size of the source ROI in the luminance plane.
<i>pSrcCb</i>	Pointer to the origin of the source ROI in Cb chrominance plane.
<i>srcCbStep</i>	Width in bytes of the source image Cb chrominance plane.
<i>pSrcCr</i>	Pointer to the origin of the source image ROI in Cr chrominance plane.
<i>srcCrStep</i>	Width in bytes of the source image Cr chrominance plane.
<i>pDstY</i>	Pointer to the origin of the destination image ROI in the luminance plane.
<i>dstYStep</i>	Width in bytes of the destination image luminance plane.
<i>yDstRoiSize</i>	Size of the destination ROI in the luminance plane.
<i>pDstCb</i>	Pointer to the origin of the destination image ROI in Cb chrominance plane.
<i>dstCbStep</i>	Width in bytes of the destination image Cb chrominance plane.
<i>pDstCr</i>	Pointer to the origin of the destination image ROI in Cr chrominance plane.
<i>dstCrStep</i>	Width in bytes of the destination image Cr chrominance plane.
<i>warpParams</i>	Array of warping parameters – 4 pairs of motion vectors, describing, in the order they are stored in the array, how the upper left, upper right, lower left, and lower right corners of the destination ROI are mapped onto the source image.
<i>wda</i>	Warping displacement accuracy flag, if set to 0, pixel displacements are quantized to half-pixel accuracy, otherwise – to 1/16-pixel accuracy.
<i>rounding</i>	Rounding value that is used in pixel interpolation, can be 0 or 1.
<i>fillMode</i>	Flag that defines the fill-mode action for the values of the source pixels for which the calculated location in the source image lies outside of the source image ROI. This parameter takes one of the following values:

0, indicating color fill mode, the “outside” Y, Cb, and Cr pixel values are set to *fillColor*[0], *fillColor*[1], and *fillColor*[2], respectively.

1 – *black* fill mode, the “outside” pixel values are set as follows: Y = 16, Cb = Cr = 128.

2 – *gray* fill mode, the “outside” pixel values are all set to 128.

3 – *clip* fill mode, the “outside” pixel values are extrapolated from the values of pixels at the ROI border, as specified in [\[ITUH263\]](#), Annex D.

*fillColor*                      Array of fill color values used in color fill mode.

## Description

The function `ippiResample_H263_8u_P3R` is declared in the `ippvc.h` header file. This function resamples a YCbCr picture as specified in [\[ITUH263\]](#), Annex P. The destination picture ROI is mapped onto the source picture ROI as defined by *warpParams*. The pixels falling outside the source image ROI are estimated according to *fillMode*.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>yRoiSize</i> or <i>yDstRoiSize</i> have a field that is odd or less than 4.

---

## UpsampleFour\_H263

*Performs factor-of-4 picture upsampling.*

---

## Syntax

```
IppStatus ippiUpsampleFour_H263_8u_C1R(const Ipp8u *pSrc, int srcStep, IppiSize
    srcRoiSize, Ipp8u *pDst, int dstStep, int rounding, int fillColor);
```

## Parameters

*pSrc*                              Pointer to the origin of the source image region of interest (ROI).



---

<i>srcStep</i>	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source image.
<i>srcRoiSize</i>	Size of the source ROI.
<i>pDst</i>	Pointer to the origin of the destination image ROI.
<i>dstStep</i>	Width in bytes of the destination image plane.
<i>rounding</i>	Rounding value used in pixel interpolation, can be 0 or 1.
<i>fillColor</i>	Fill color value used for the source pixels for which the calculated location in the source image lies outside of the source image ROI. When <i>fillColor</i> is negative, <i>clip</i> fill-mode action is employed – the “outside” pixel values are extrapolated from the values of pixels at the ROI border, as specified in [ITUH263], Annex D.

### Description

The function `ippiUpsampleFour_H263_8u_C1R` is declared in the `ippvc.h` header file. This function performs factor-of-4 picture upsampling as specified in [ITUH263], Annex P, subclause P.5.1.

This function is used in the H.263 decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> has a field that is odd or less than 4.

---

## DownsampleFour\_H263

*Performs factor-of-4 picture downsampling.*

---

### Syntax

```
IppStatus ippiDownsampleFour_H263_8u_C1R(const Ipp8u *pSrc, int srcStep,
    IppiSize srcRoiSize, Ipp8u *pDst, int dstStep, int rounding);
```

## Parameters

<i>pSrc</i>	Pointer to the origin of the source image region of interest (ROI).
<i>srcStep</i>	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source image.
<i>srcRoiSize</i>	Size of the source ROI.
<i>pDst</i>	Pointer to the origin of the destination image ROI.
<i>dstStep</i>	Width in bytes of the destination image plane.
<i>rounding</i>	Rounding value used in pixel interpolation, can be 0 or 1.

## Description

The function `ippiDownsampleFour_H263_8u_C1R` is declared in the `ippvc.h` header file. This function performs factor-of-4 picture downsampling as specified in [ITUH263], Annex P, subclause P.5.2.

This function is used in the H.263 decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> has a field with zero or negative value.

---

## UpsampleFour8x8\_H263

*Performs factor-of-4 upsampling on an 8x8 block.*

---

## Syntax

```
IppStatus ippiUpsampleFour8x8_H263_16s_C1R(const Ipp16s *pSrc, int srcStep,
      Ipp16s *pDst, int dstStep);
```

## Parameters

<i>pSrc</i>	Pointer to the origin of the source 8x8 block.
<i>srcStep</i>	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source block.
<i>pDst</i>	Pointer to the origin of the destination 16x16 block.
<i>dstStep</i>	Width in bytes of the destination image plane.

## Description

The function `ippiUpsampleFour8x8_H263_16s_C1R` is declared in the `ippvc.h` header file. This function performs factor-of-4 upsampling of an 8x8 source block to a 16x16 destination block, as specified in [ITUH263], Annex Q, clause Q.6.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## SpatialInterpolation\_H263

*Interpolates a picture by a factor of two horizontally, vertically, or both horizontally and vertically.*

## Syntax

```
IppStatus ippiSpatialInterpolation_H263_8u_C1R(const Ipp8u *pSrc, int srcStep,
        IppiSize srcRoiSize, Ipp8u *pDst, int dstStep, int interpType);
```

<i>pSrc</i>	Pointer to the origin of the source image region of interest (ROI).
<i>srcStep</i>	Width in bytes of the source image plane, that is, distance in bytes between the starting ends of consecutive lines of the source image.
<i>srcRoiSize</i>	Size of the source ROI.
<i>pDst</i>	Pointer to the origin of the destination image ROI.
<i>dstStep</i>	Width in bytes of the destination image plane.

*interpType* Interpolation type, takes one of the following values:  
 IPPVC\_INTERP\_HORIZONTAL, indicating one-dimensional (1-D) horizontal interpolation,  
 IPPVC\_INTERP\_VERTICAL, indicating 1-D vertical interpolation,  
 IPPVC\_INTERP\_2D, indicating 2-D interpolation.  
 See the corresponding enumerator on [p.16-3](#).

## Description

The function `ippiSpatialInterpolation_H263_8u_C1R` is declared in the `ippvc.h` header file. This function performs picture interpolation for 1-D or 2-D spatial scalability, as specified in [\[ITUH263\]](#), Annex O, clause O.6. Depending on *interpType*, the source image ROI is interpolated by a factor of 2 either horizontally, vertically, or both horizontally and vertically, the size of the destination image ROI being equal to the size of the source image ROI multiplied by 2 in the direction(s) for which the interpolation is performed.

This function is used in the H.263 decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsSizeErr</code>	Indicates an error condition if <i>srcRoiSize</i> has a field that is odd or less than 4.

---

## Boundary Filtering

---

### FilterBlockBoundaryHorEdge\_H263, FilterBlockBoundaryVerEdge\_H263

*Perform block boundary filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.*

---

#### Syntax

```
IppStatus ippiFilterBlockBoundaryHorEdge_H263_8u_C1IR(Ipp8u *pSrcDst, int  
    srcDstStep);  
IppStatus ippiFilterBlockBoundaryVerEdge_H263_8u_C1IR(Ipp8u *pSrcDst, int  
    srcDstStep);
```

#### Parameters

<i>pSrcDst</i>	Pointer to the origin of the lower (HorEdge) or the right (VerEdge) 16x16 block.
<i>srcDstStep</i>	Width in bytes of the image plane.

#### Description

The functions `ippiFilterBlockBoundaryHorEdge_H263_8u_C1IR` and `ippiFilterBlockBoundaryVerEdge_H263_8u_C1IR` are declared in the `ippvc.h` header file. These functions perform block boundary filtering on bordering edges, horizontal and vertical respectively, of two adjacent 16x16 blocks, as specified in [\[ITUH263\]](#), Annex Q, subclause Q.7.1.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrcDst</i> is NULL.

---

## FilterDeblocking8x8HorEdge\_H263, FilterDeblocking8x8VerEdge\_H263

*Perform deblocking filtering of one block edge on the reconstructed frames.*

---

### Syntax

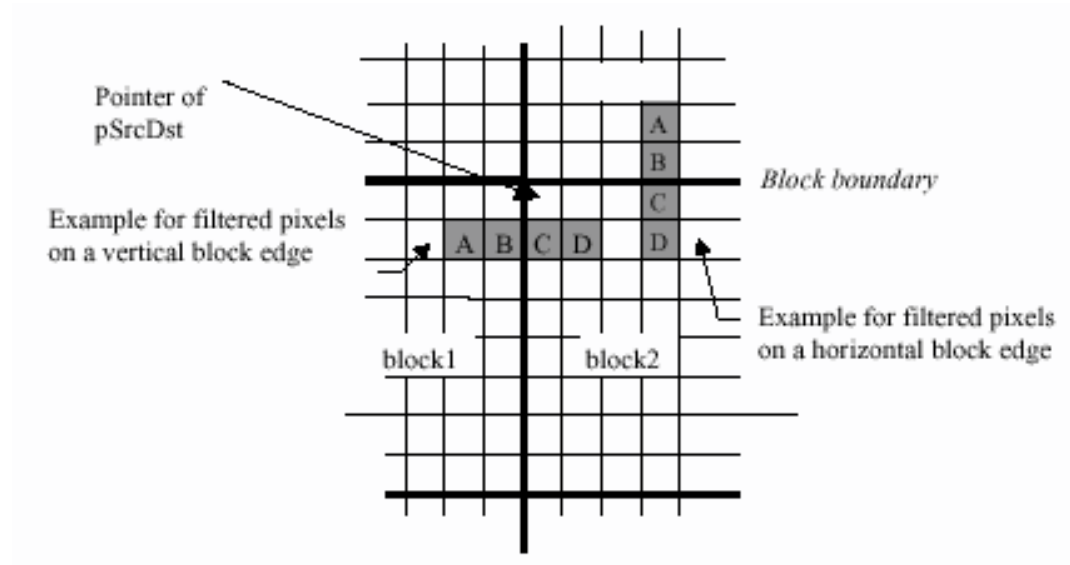
```
IppStatus ippiFilterDeblocking8x8HorEdge_H263_8u_C1IR(Ipp8u* pSrcDst,  
    int srcDstStep, int QP);  
IppStatus ippiFilterDeblocking8x8VerEdge_H263_8u_C1IR(Ipp8u* pSrcDst,  
    int srcDstStep, int QP);
```

### Parameters

<i>pSrcDst</i>	Pointer to the first pixel of the second block (block 2) of the two applied blocks.
<i>srcDstStep</i>	Width of the source and destination plane.
<i>QP</i>	Quantization parameter. The value of <i>QP</i> is found as described in Section J.3 of Annex J/H.263+.

### Description

The functions `ippiFilterDeblocking8x8HorEdge_H263_8u_C1IR` and `ippiFilterDeblocking_VerEdge8x8H263_8u_C1IR` are declared in the `ippvc.h` file. These functions perform deblocking filtering of one block edge (horizontal or vertical, respectively) on the reconstructed frames. The pointer *pSrcDst* points to the first pixel of the block 2 as shown in [Figure 16-47](#).

**Figure 16-47    Deblocking Filtering Layout**

These functions are used in the H.263 encoder and decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>pSrcDst</i> pointer is NULL.
<code>ippStsQPErr</code>	Indicates an error condition if the quantizer value has a zero or negative value, or if it is greater than 31.

---

## FilterDeblocking16x16HorEdge\_H263, FilterDeblocking16x16VerEdge\_H263

*Perform deblocking filtering on a horizontal or vertical boundary of two adjacent 16x16 blocks.*

---

### Syntax

```
IppStatus ippiFilterDeblocking16x16HorEdge_H263_8u_C1IR(Ipp8u *pSrcDst, int  
srcDstStep);
```

```
IppStatus ippiFilterDeblocking16x16VerEdge_H263_8u_C1IR(Ipp8u *pSrcDst, int  
srcDstStep);
```

### Parameters

*pSrcDst*                      Pointer to the origin of the lower (HorEdge) or the right (VerEdge) 16x16 block.

*step*                        Width in bytes of the image plane.

### Description

The functions `ippiFilterDeblocking16x16HorEdge_H263_8u_C1IR` and `ippiFilterDeblocking16x16VerEdge_H263_8u_C1IR` are declared in the `ippvc.h` header file. These functions perform deblocking filtering on bordering edges, horizontal and vertical respectively, of two adjacent 16x16 blocks, as specified in [\[ITUH263\]](#), Annex Q, subclause Q.7.2 and Annex J, clause J.3.

### Return Values

`ippStsNoErr`                      Indicates no error.

`ippStsNullPtrErr`                Indicates an error condition if *pSrcDst* is NULL.



## Middle Level Functions

---

### ReconstructCoeffsIntra\_H263

*Reconstructs DCT coefficients for an intra coded block.*

---

#### Syntax

```
IppStatus ippiReconstructCoeffsIntra_H263_1u16s(Ipp8u **ppBitStream, int
    *pBitOffset, Ipp16s *pCoef, int *pIndxLastNonZero, int cbp, int QP, int
    advIntraFlag, int scan, int modQuantFlag);
```

#### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>cbp</i>	Coded block pattern, when set to 0 indicates that the block contains only Intra DC coefficient.
<i>QP</i>	Quantization parameter.
<i>advIntraFlag</i>	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
<i>scan</i>	Type of the inverse scan, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating alternate-vertical scan.  See the corresponding enumerator on <a href="#">p.16-3</a> .

<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.
---------------------	---

## Description

The function `ippiReconstructCoeffsIntra_H263_1u16s` is declared in the `ippvc.h` header file. This function performs decoding, dequantization, and inverse scan of the DCT coefficients for one Intra coded block.

Intra DC decoding process is specified in [ITUH263], subclause 5.4.1. Intra AC VLC decoding process is specified in [ITUH263], subclause 5.4.2, and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. When in Advanced Intra Coding mode, VLC Table I.2 from [ITUH263] Annex I is used for all Intra DC and Intra AC coefficients, otherwise Table 16 [ITUH263] is used to decode AC coefficients only (for blocks with non-zero *cbp*). When not in Advanced Intra Coding mode, the dequantization processes for the Intra DC and for all other non-zero coefficients are specified in [ITUH263], subclause 6.2.1, otherwise all the coefficients are dequantized as specified in [ITUH263] Annex I, clause I.3. When not in Advanced Intra Coding mode and not in Modified Quantization mode, the output coefficients other than the Intra DC one are clipped to the range [-2048, 2047] ([ITUH263], subclause 6.2.2).

The DCT coefficients, encoded in the bitstream in the classical zigzag, alternate-horizontal, or alternate-vertical scan order, are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output. The three scan patterns are shown in [ITUH263], Figure 14 and [ITUH263], Annex I, Figure I.2.

This function is used in the H.263 and MPEG-4 decoders included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCErr</code>	Indicates an error condition if an illegal code is detected through the stream processing.
<code>ippStsQPErr</code>	Indicates an error condition if <i>QP</i> is out of the range [1, 31].

---

## ReconstructCoeffsInter\_H263

*Reconstructs DCT coefficients for an inter coded block.*

---

### Syntax

```
IppStatus ippiReconstructCoeffsInter_H263_1u16s(Ipp8u **ppBitStream, int
    *pBitOffset, Ipp16s *pCoef, int *pIndxLastNonZero, int QP, int
    modQuantFlag);
```

### Parameters

<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>pCoef</i>	Pointer to the output coefficients.
<i>pIndxLastNonZero</i>	Pointer to the index of the last non-zero coefficient in the scanning order. If an error is detected while decoding a coefficient, the index of the last decoded coefficient is returned in <i>*pIndxLastNonZero</i> . If the block has no correctly decoded coefficients, <i>*pIndxLastNonZero</i> is set to -1.
<i>QP</i>	Quantization parameter.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

### Description

The function `ippiReconstructCoeffsInter_H263_1u16s` is declared in the `ippvc.h` header file. This function performs decoding, dequantization, and inverse scan of the DCT coefficients for one Inter coded block.

Inter DCT VLC decoding process is specified in [ITUH263], subclause 5.4.2 (Table 16), and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. The dequantization process is specified in [ITUH263], subclause 6.2.1. When not in Modified Quantization mode, the output coefficients are clipped to the range [-2048, 2047] ([ITUH263], subclause 6.2.2). The DCT coefficients, encoded in the bitstream in the classical zigzag scan order ([ITUH263], Figure 14), are reordered in the function into the normal order, that is, the order in which the coefficients are arranged on DCT output.

This function is used in the H.263 and MPEG-4 decoders included into IPP Samples. See [introduction](#) to this section.

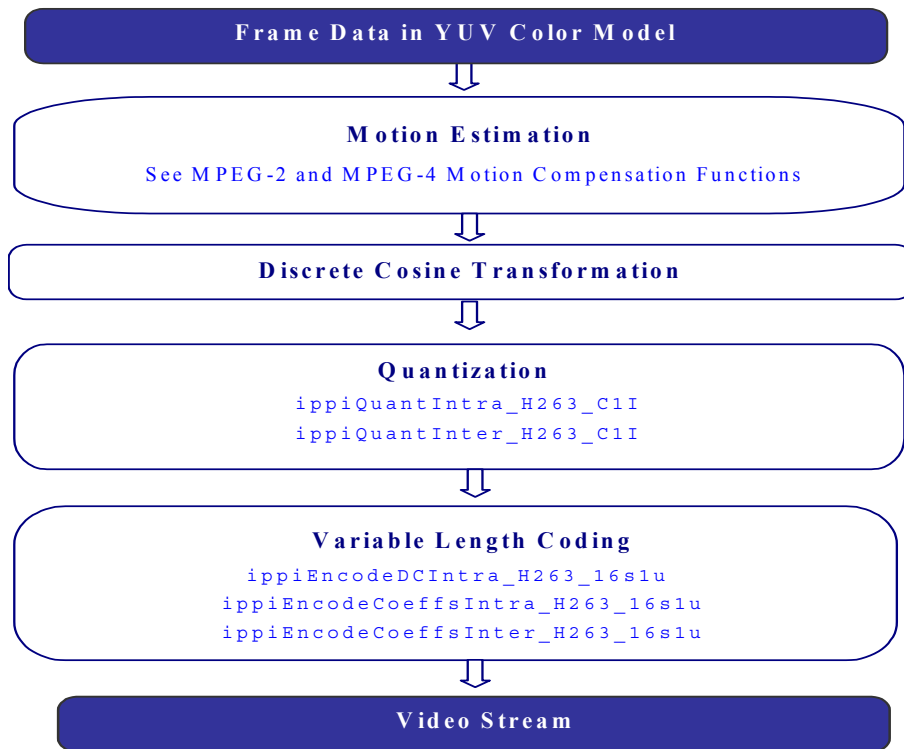
### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsVLCErr</code>	Indicates an error condition if an illegal code is detected through the stream processing.
<code>ippStsQPErr</code>	Indicates an error condition if <i>QP</i> is out of the range [1, 31].

## H.263 Encoder Functions

This section describes the main steps of H.263 ([[ITUH263](#)]) video encoding in accordance with the pipeline shown in [Figure 16-48](#).

**Figure 16-48** H.263 Encoding Pipeline



## VLC Encoding

---

### EncodeDCIntra\_H263

*Encodes and puts a quantized DC coefficient for an intra coded block into bitstream.*

---

#### Syntax

```
IppStatus ippiEncodeDCIntra_H263_16s1u(Ipp16s qDC, Ipp8u **ppBitStream, int
    *pBitOffset);
```

#### Parameters

<i>qDC</i>	Quantized DC coefficient.
<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. * <i>ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by * <i>ppBitStream</i> . Valid within the range 0 to 7. * <i>pBitOffset</i> is updated by the function.

#### Description

The function `ippiEncodeDCIntra_H263_16s1u` is declared in the `ippvc.h` header file. This function performs fixed length encoding of the DC coefficient for one Intra coded block and puts the code into the bitstream. Intra DC encoding process is specified in [ITUH263], subclause 5.4.1.

This function is used in the H.263 and MPEG-4 encoders included into IPP Samples. See [introduction](#) to H.263 section.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if * <i>pBitOffset</i> is out of the range [0, 7].

## EncodeCoeffsIntra\_H263

Encodes and puts quantized DCT coefficients for an intra coded block into bitstream.

### Syntax

```
IppStatus ippiEncodeCoeffsIntra_H263_16s1u(Ipp16s *pQCoef, Ipp8u **ppBitStream,
int *pBitOffset, int countNonZero, int advIntraFlag, int modQuantFlag, int
scan);
```

### Parameters

<i>pQCoef</i>	Pointer to the array of quantized DCT coefficients. <i>pQCoef</i> [0] is the DC coefficient.
<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>countNonZero</i>	Number of non-zero coefficients in the block. Valid within the range 1 to 64.
<i>advIntraFlag</i>	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.
<i>scan</i>	Type of the scan to be performed on the coefficients before encoding, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_HORIZONTAL, indicating the alternate-horizontal scan, IPPVC_SCAN_VERTICAL, indicating the alternate-vertical scan, IPPVC_SCAN_NONE, indicating that no scan is to be performed, that is, the input coefficients are already in the scan order.  See the corresponding enumerator on <a href="#">p.16-3</a> .

### Description

The function `ippiEncodeCoeffsIntra_H263_16s1u` is declared in the `ippvc.h` header file. This function performs VLC encoding of the quantized AC coefficients in a scan order for one Intra coded block and puts the codes into the bitstream. Intra AC VLC encoding process is specified in [ITUH263], subclause 5.4.2, and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use. When in Advanced Intra Coding mode, VLC Table I.2 from [ITUH263] Annex I is used for all Intra DC and Intra AC coefficients, otherwise Table 16 [ITUH263] is used to encode AC coefficients (starting from `pQCoef[1]`) only. In this case `countNonZero` is the number of non-zero AC coefficients.

This function is used in the H.263 and MPEG-4 encoders included into IPP Samples. See [introduction](#) to H.263 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <code>*pBitOffset</code> is out of the range [0, 7].
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <code>countNonZero</code> is out of the range [1, 64].

---

## EncodeCoeffsInter\_H263

*Encodes and puts quantized DCT coefficients for inter coded block into bitstream.*

---

### Syntax

```
IppStatus ippiEncodeCoeffsInter_H263_16s1u(Ipp16s *pQCoef, Ipp8u **ppBitStream,  
int *pBitOffset, int countNonZero, int modQuantFlag, int scan);
```

### Parameters

<code>pQCoef</code>	Pointer to the array of quantized DCT coefficients. <code>pQCoef[0]</code> is the DC coefficient.
---------------------	---



<i>ppBitStream</i>	Pointer to pointer to the current byte in the bitstream buffer. <i>*ppBitStream</i> is updated by the function.
<i>pBitOffset</i>	Pointer to the bit position in the byte pointed by <i>*ppBitStream</i> . Valid within the range 0 to 7. <i>*pBitOffset</i> is updated by the function.
<i>countNonZero</i>	Number of non-zero coefficients in the block. Valid within the range 1 to 64.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.
<i>scan</i>	Type of the scan to be performed on the coefficients before encoding, takes one of the following values: IPPVC_SCAN_ZIGZAG, indicating the classical zigzag scan, IPPVC_SCAN_NONE, indicating that no scan is to be performed, that is, the input coefficients are already in the scan order.  See the corresponding enumerator on <a href="#">p.16-3</a> .

## Description

The function `ippiEncodeCoeffsInter_H263_16s1u` is declared in the `ippvc.h` header file. This function performs VLC encoding of the quantized DCT coefficients in a scan order for one Inter coded block and puts the codes into the bitstream. Inter DCT VLC encoding process is specified in [ITUH263], subclause 5.4.2 (Table 16), and is modified as specified in [ITUH263] Annex T, clause T.4, when Modified Quantization mode is in use.

This function is used in the H.263 and MPEG-4 encoders included into IPP Samples. See [introduction](#) to H.263 section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsBitOffsetErr</code>	Indicates an error condition if <i>*pBitOffset</i> is out of the range [0, 7].
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <i>countNonZero</i> is out of the range [1, 64].

## Quantization

---

### QuantIntra\_H263

*Performs quantization on an intra coded block.*

---

#### Syntax

```
IppStatus ippiQuantIntra_H263_16s_C1I(Ipp16s *pSrcDst, int QP, int  
    *pCountNonZero, int advIntraFlag, int modQuantFlag);
```

#### Parameters

<i>pSrcDst</i>	Pointer to the coefficient buffer of the block.
<i>QP</i>	Quantization parameter.
<i>pCountNonZero</i>	Pointer to the number of non-zero coefficients after quantization.
<i>advIntraFlag</i>	Flag equal to a non-zero value when Advanced Intra Coding mode is in use, equal to 0 otherwise.
<i>modQuantFlag</i>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

#### Description

The function `ippiQuantIntra_H263_16s_C1I` is declared in the `ippvc.h` header file. This function performs quantization on an Intra coded block according to H.263 standard. The standard specifies dequantization process, while quantization decision levels are not defined. When not in Advanced Intra Coding mode, the Intra DC coefficient is dequantized using uniformly placed reconstruction levels with a step size of 8, and the other DCT coefficients are reconstructed using equally spaced levels with a central dead-zone around zero and with a step size of  $2 \cdot QP$  ([ITUH263], subclauses 4.2.4, 6.2). When in Advanced Intra Coding mode, all the block coefficients are dequantized using a reconstruction spacing without a dead-zone and with a step size of  $2 \cdot QP$  ([ITUH263] Annex I, clause I.3). When not in Modified Quantization mode, the quantized Intra DC coefficient (when not in Advanced Intra Coding mode) is clipped to the range [1, 254], and the other quantized coefficients (all coefficients, if in Advanced Intra Coding mode) are clipped to the range [-127, 127].

This function is used in the H.261, H.263, and MPEG-4 encoders included into IPP Samples. See [introduction](#) to H.263 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsQPErr</code>	Indicates an error condition if $QP$ is out of the range $[1, 31]$ .

---

## QuantInter\_H263

*Performs quantization on an inter coded block.*

---

### Syntax

```
IppStatus ippQuantInter_H263_16s_C1I(Ipp16s *pSrcDst, int QP, int  
    *pCountNonZero, int modQuantFlag);
```

### Parameters

<code>pSrcDst</code>	Pointer to the coefficient buffer of the block.
<code>QP</code>	Quantization parameter.
<code>pCountNonZero</code>	Pointer to the number of non-zero coefficients after quantization.
<code>modQuantFlag</code>	Flag equal to a non-zero value when Modified Quantization mode is in use, equal to 0 otherwise.

### Description

The function `ippQuantInter_H263_16s_C1I` is declared in the `ippvc.h` header file. This function performs quantization on an Inter coded block according to H.263 standard. The standard specifies dequantization process, while quantization decision levels are not defined. The DCT coefficients are reconstructed using equally spaced levels with a central dead-zone around zero and with a step size of  $2 * QP$  ([[ITUH263](#)], subclauses 4.2.4, 6.2). When not in Modified Quantization mode, the quantized coefficients are clipped to the range  $[-127, 127]$ .

This function is used in the H.261, H.263, and MPEG-4 encoders included into IPP Samples. See [introduction](#) to H.263 section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
--------------------------	---------------------

<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsQPErr</code>	Indicates an error condition if <i>QP</i> is out of the range [1, 31].

## Resampling

---

### DownsampleFour16x16\_H263

*Performs factor-of-4 downsampling on a 16x16 block.*

---

#### Syntax

```
IppStatus ippDownsampleFour16x16_H263_16s_C1R(const Ipp16s *pSrc, int srcStep,
        Ipp16s* pDst, int dstStep);
```

#### Parameters

<i>pSrc</i>	Pointer to the origin of the source 16x16 block.
<i>srcStep</i>	Width in bytes of the source image plane, that is, distance in bytes between the starts of consecutive lines of the source block.
<i>pDst</i>	Pointer to the origin of the destination 8x8 block.
<i>dstStep</i>	Width in bytes of the destination image plane.

#### Description

The function `ippDownsampleFour16x16_H263_16s_C1R` is declared in the `ippvc.h` header file. This function performs factor-of-4 downsampling of a 16x16 source block to an 8x8 destination block, which is used for block encoding in Reduced-Resolution Update mode specified in [ITUH263], Annex Q. The inverse factor-of-4 upsampling procedure, specified in [ITUH263], Annex Q, clause Q.6, is performed by the function [UpsampleFour8x8\\_H263](#).

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## H.264

This section describes ippVC functions for decoding of video data in accordance with JVT-G050 ([JVTG050]) and ITUH264 [ITUH264] standards.

The use of some functions described in this section is demonstrated in Intel® IPP Samples downloadable from <http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm> .

The following enumeration is used to indicate prediction modes of the Intra\_4x4 prediction process for luma samples (8.3.1 of JVT-G050):

```
typedef enum {    IPP_4x4_VERT      = 0,
                  IPP_4x4_HOR      = 1,
                  IPP_4x4_DC       = 2,
                  IPP_4x4_DIAG_DL  = 3,
                  IPP_4x4_DIAG_DR  = 4,
                  IPP_4x4_VR       = 5,
                  IPP_4x4_HD       = 6,
                  IPP_4x4_VL       = 7,
                  IPP_4x4_HU       = 8
                } IppIntra4x4PredMode_H264;
```

[Table 16-22](#) shows the correspondence between prediction modes and constants of enum IppIntra4x4PredMode\_H264.

Table 16-22

Name of Constant	Prediction Mode	Chapter in JVT-G050
IPP_4x4_VERT	Intra_4x4_Vertical	8.3.1.2.1
IPP_4x4_HOR	Intra_4x4_Horizontal	8.3.1.2.2
IPP_4x4_DC	Intra_4x4_DC	8.3.1.2.3
IPP_4x4_DIAG_DL	Intra_4x4_Diagonal_Down_Left	8.3.1.2.4
IPP_4x4_DIAG_DR	Intra_4x4_Diagonal_Down_Right	8.3.1.2.5
IPP_4x4_VR	Intra_4x4_Vertical_Right	8.3.1.2.6
IPP_4x4_HD	Intra_4x4_Horizontal_Down	8.3.1.2.7
IPP_4x4_VL	Intra_4x4_Vertical_Left	8.3.1.2.8
IPP_4x4_HU	Intra_4x4_Horizontal_Up	8.3.1.2.9

The following enumeration is used to indicate prediction modes of the Intra\_16x16 prediction process for luma samples (8.3.2 of JVT-G050):

```
typedef enum {    IPP_16X16_VERT      = 0,
                  IPP_16X16_HOR      = 1,
                  IPP_16X16_DC       = 2,
                  IPP_16X16_PLANE    = 3,
                  } IppIntra16x16PredMode_H264;
```

[Table 16-23](#) shows the correspondence between prediction modes and constants of enum IppIntra16x16PredMode\_H264.

**Table 16-23**

Name of Constant	Prediction Mode	Chapter in JVT-G050
IPP_16X16_VERT	Intra_16x16_Vertical	8.3.2.1
IPP_16X16_HOR	Intra_16x16_Horizontal	8.3.2.2
IPP_16X16_DC	Intra_16x16_DC	8.3.2.3
IPP_16X16_PLANE	Intra_16x16_Plane	8.3.2.4

The following enumeration is used to indicate prediction modes of the Intra prediction process for chroma samples (8.3.3 of JVT-G050):

```
typedef enum {    IPP_CHROMA_DC      = 0,
                  IPP_CHROMA_HOR     = 1,
                  IPP_CHROMA_VERT    = 2,
                  IPP_CHROMA_PLANE   = 3,
                  } IppIntraChromaPredMode_H264;
```

[Table 16-24](#) shows the correspondence between prediction modes and constants of enum IppIntraChromaPredMode\_H264.

**Table 16-24**

Name of Constant	Prediction Mode	Chapter in JVT-G050
IPP_CHROMA_DC	Intra_Chroma_DC	8.3.3.1
IPP_CHROMA_HOR	Intra_Chroma_Horizontal	8.3.3.2
IPP_CHROMA_VERT	Intra_Chroma_Vertical	8.3.3.3
IPP_CHROMA_PLANE	Intra_Chroma_Plane	8.3.3.4

The following enumeration is used to indicate availability of the corresponding positions for prediction in the picture:

```
typedef enum { IPP_UPPER          = 1,
               IPP_LEFT           = 2,
               IPP_CENTER         = 4,
               IPP_RIGHT          = 8,
               IPP_LOWER          = 16,
               IPP_UPPER_LEFT     = 32,
               IPP_UPPER_RIGHT    = 64,
               IPP_LOWER_LEFT     = 128,
               IPP_LOWER_RIGHT    = 256,
               } IppLayoutFlag;
```

The following enumeration is used to indicate image type of the picture:

```
typedef enum _IPPVC_FRAME_FIELD_FLAG
{
    IPPVC_FRAME          = 0x0,
    IPPVC_TOP_FIELD      = 0x1,
    IPPVC_BOTTOM_FIELD   = 0x2
} IPPVC_FRAME_FIELD_FLAG;
```

Table 16-25     H.264 Video Decoder Functions

Function Short Name	Description
<b>CAVLC Parsing</b>	
<a href="#">DecodeCAVLCCoeffs_H264</a>	Decodes any non-Chroma DC coefficients CAVLC coded.
<a href="#">DecodeCAVLCChromaDcCoeffs_H264</a>	Decodes Chroma DC coefficients CAVLC coded.
<a href="#">DecodeExpGolombOne_H264</a>	Decodes one Exp-Golomb code accordance to 9.1 H.264 standard.
<b>Inverse Quantization and Inverse Transform</b>	
<a href="#">TransformDequantLumaDC_H264</a>	Performs integer inverse transformation and dequantization for 4x4 luma DC coefficients.

**Table 16-25 H.264 Video Decoder Functions (continued)**

Function Short Name	Description
<a href="#"><u>TransformDequantChromaDC_H264</u></a>	Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients.
<a href="#"><u>DequantTransformResidual_H264</u></a>	Performs dequantization, integer inverse transformation, and shift for a 4x4 block of residuals.
<a href="#"><u>DequantTransformResidualAndAdd_H264</u></a>	Performs dequantization, integer inverse transformation, shift for a 4x4 block of residuals with subsequent intra prediction or motion compensation.
<a href="#"><u>TransformPrediction_H264</u></a>	Performs inverse transform of inter prediction samples for the current macroblock in decoding process for P macroblocks in SP slices or SI macroblocks.
<a href="#"><u>DequantTransformResidual_SISP_H264</u></a>	Performs integer inverse transformation and dequantization of one block in P macroblocks in SP slices or SI macroblocks.
<a href="#"><u>TransformDequantChromaDC_SISP_H264</u></a>	Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients in P macroblocks in SP slices or SI macroblocks.
<b>Intra Prediction</b>	
<a href="#"><u>PredictIntra_4x4_H264</u></a>	Performs luma component prediction for Intra 4x4 macroblock type.
<a href="#"><u>PredictIntra_16x16_H264</u></a>	Performs luma component prediction for Intra 16x16 macroblock type.
<a href="#"><u>PredictIntraChroma8x8_H264</u></a>	Performs chroma component prediction for Intra macroblock type.
<b>Inter Prediction</b>	
<a href="#"><u>ExpandPlane_H264</u></a>	Expands the plane.
<a href="#"><u>InterpolateLuma_H264</u></a>	Performs interpolation for motion estimation of the luma component.
<a href="#"><u>InterpolateLumaTop_H264</u></a>	Performs interpolation for motion estimation of the luma component at the frame top boundary.



**Table 16-25 H.264 Video Decoder Functions (continued)**

Function Short Name	Description
<a href="#"><u>InterpolateLumaBottom_H264</u></a>	Performs interpolation for motion estimation of the luma component at the frame bottom boundary.
<a href="#"><u>InterpolateChroma_H264</u></a>	Performs interpolation for motion estimation of the chroma component.
<a href="#"><u>InterpolateChromaTop_H264</u></a>	Performs interpolation for motion estimation of the chroma component at the frame top boundary.
<a href="#"><u>InterpolateChromaBottom_H264</u></a>	Performs interpolation for motion estimation of the chroma component at the frame bottom boundary.
<a href="#"><u>InterpolateBlock_H264</u></a>	Calculates the average value for each source pair values in block.
<a href="#"><u>WeightedAverage_H264</u></a>	Averages two blocks with weights.
<a href="#"><u>UniDirWeightBlock_H264</u></a>	Weights source block.
<a href="#"><u>BiDirWeightBlock_H264</u></a>	Averages two blocks with two weights and two offsets.
<a href="#"><u>BiDirWeightBlockImplicit_H264</u></a>	Averages two blocks with weights if <i>uLog2wd</i> is equal to 5.
<b>Macroblock Reconstruction</b>	
<a href="#"><u>ReconstructChromaInterMB_H264</u></a>	Reconstructs Inter Chroma macroblock.
<a href="#"><u>ReconstructChromaIntraHalvesMB_H264</u></a>	Reconstructs two halves of Intra Chroma macroblock.
<a href="#"><u>ReconstructChromaIntraMB_H264</u></a>	Reconstructs Intra Chroma macroblock.
<a href="#"><u>ReconstructChromaInter4x4MB_H264</u></a>	Reconstructs 4X4 Inter Chroma macroblock for high profile.
<a href="#"><u>ReconstructChromaIntraHalves4x4MB_H264</u></a>	Reconstructs two halves of 4X4 Intra Chroma macroblock for high profile.
<a href="#"><u>ReconstructChromaIntra4x4MB_H264</u></a>	Reconstructs 4X4 Intra Chroma macroblock for high profile.
<a href="#"><u>ReconstructLumaInterMB_H264</u></a>	Reconstructs Inter Luma macroblock.
<a href="#"><u>ReconstructLumaIntraHalfMB_H264</u></a>	Reconstructs half of Intra Luma macroblock.
<a href="#"><u>ReconstructLumaIntraMB_H264</u></a>	Reconstructs Intra Luma macroblock.

**Table 16-25 H.264 Video Decoder Functions (continued)**

Function Short Name	Description
<a href="#"><u>ReconstructLumaInter4x4MB_H264</u></a>	Reconstructs 4X4 Inter Luma macroblock for high profile.
<a href="#"><u>ReconstructLumaIntraHalfMB_H264</u></a>	Reconstructs half of 4X4 Intra Luma macroblock for high profile.
<a href="#"><u>ReconstructLumaIntra4x4MB_H264</u></a>	Reconstructs 4X4 Intra Luma macroblock for high profile.
<a href="#"><u>ReconstructLumaInter8x8MB_H264</u></a>	Reconstructs 8X8 Inter Luma macroblock for high profile.
<a href="#"><u>ReconstructLumaIntraHalf8x8MB_H264</u></a>	Reconstructs half of 8X8 Intra Luma macroblock for high profile.
<a href="#"><u>ReconstructLumaIntra8x8MB_H264</u></a>	Reconstructs 8X8 Intra Luma macroblock for high profile.
<a href="#"><u>ReconstructLumaIntra16x16MB_H264</u></a>	Reconstructs Intra 16X16 Luma macroblock.
<a href="#"><u>ReconstructLumaIntra_16x16MB_H264</u></a>	Reconstructs Intra 16X16 Luma macroblock for high profile.
<b>Deblocking Filtering</b>	
<a href="#"><u>FilterDeblockingLuma_VerEdge_H264</u></a>	Performs deblocking filtering on the vertical edges of the luma 16x16 macroblock.
<a href="#"><u>FilterDeblockingLuma_VerEdge_MBAFF_H264</u></a>	Performs deblocking filtering on the external vertical edges of half of 16x16 luma macroblock.
<a href="#"><u>FilterDeblockingLuma_HorEdge_H264</u></a>	Performs deblocking filtering on the horizontal edges of the luma 16x16 macroblock.
<a href="#"><u>FilterDeblockingChroma_VerEdge_H264</u></a>	Performs deblocking filtering on the vertical edges of the chroma 8x8 macroblock.
<a href="#"><u>FilterDeblockingChroma_VerEdge_MBAFF_H264</u></a>	Performs deblocking filtering on the external vertical edges of half of 8x8 chroma macroblock.
<a href="#"><u>FilterDeblockingChroma_HorEdge_H264</u></a>	Perform deblocking filtering on the chroma 8x8 macroblock.

Table 16-26 H.264 Video Encoder Functions

Function Short Name	Description
<b>Edges Detection</b>	
<a href="#">EdgesDetect16x16</a>	Detects edges inside a 16X16 block.
<b>Calculation of Inter Predicted Blocks</b>	
<a href="#">InterpolateLuma H264</a>	Performs interpolation for motion estimation of the luma component.
<a href="#">InterpolateLumaTop H264</a>	Performs interpolation for motion estimation of the luma component at the frame top boundary.
<a href="#">InterpolateLumaBottom H264</a>	Performs interpolation for motion estimation of the luma component at the frame bottom boundary.
<a href="#">InterpolateChroma H264</a>	Performs interpolation for motion estimation of the chroma component.
<a href="#">InterpolateChromaTop H264</a>	Performs interpolation for motion estimation of the chroma component at the frame top boundary.
<a href="#">InterpolateChromaBottom H264</a>	Performs interpolation for motion estimation of the chroma component at the frame bottom boundary.
<b>Calculation of Intra Predicted Blocks</b>	
<a href="#">PredictIntra 4x4 H264</a>	Performs intra prediction for a 4x4 luma component.
<a href="#">PredictIntra 16x16 H264</a>	Performs intra prediction for a 16x16 luma component.
<a href="#">PredictIntraChroma8x8 H264</a>	Performs intra prediction for a 8x8 chroma component.
<b>Estimation of Inter and Intra Predicted Blocks</b>	
<a href="#">SAD4x4</a>	Evaluates sum of absolute difference between current and reference 4X4 blocks.
<a href="#">SAD8x8</a>	Evaluates sum of absolute difference between current and reference 8X8 blocks.
<a href="#">SAD16x16</a>	Evaluates sum of absolute difference between current and reference blocks.

**Table 16-26 H.264 Video Encoder Functions (continued)**

Function Short Name	Description
<a href="#"><u>SAD16x16Blocks8x8</u></a>	Evaluates four partial sums of absolute differences between current and reference 16X16 blocks.
<a href="#"><u>SAD16x16Blocks4x4</u></a>	Evaluates 16 partial sums of absolute differences between current and reference 16X16 blocks.
<b>Obtaining of Residual and DC Blocks</b>	
<a href="#"><u>GetDiff4x4</u></a>	Calculates a residual 4x4 block in the cases of Intra 4x4 Luma block or Inter Luma block.
<a href="#"><u>SumsDiff16x16Blocks4x4</u></a>	Calculates a 4x4 DC Luma block and 4x4 residual blocks in the case of an Intra 16x16 Luma block.
<a href="#"><u>SumsDiff8x8Blocks4x4</u></a>	Calculates a 2x2 DC Chroma block and residual 4x4 chroma blocks.
<b>Forward Transform and Quantization</b>	
<a href="#"><u>TransformQuantChromaDC_H264</u></a>	Performs forward transform and quantization for 2x2 DC Chroma blocks.
<a href="#"><u>TransformQuantLumaDC_H264</u></a>	Performs forward transform and quantization for 4x4 DC Luma blocks.
<a href="#"><u>TransformQuantResidual_H264</u></a>	Performs forward transform and quantization for 4x4 residual blocks.
<a href="#"><u>TransformLuma8x8Fwd_H264</u></a>	Performs forward 8x8 transform for 8x8 Luma blocks without normalisation.
<a href="#"><u>QuantLuma8x8_H264</u></a>	Performs quantization for 8x8 Luma block coefficients including 8x8 transform normalization.
<a href="#"><u>GenScaleLevel8x8_H264</u></a>	Generates ScaleLevel matrices for forward and inverse quantization including normalization for 8x8 forward and inverse transform.
<b>CAVLC Functions</b>	
<a href="#"><u>EncodeCoeffsCAVLC_H264</u></a>	Calculates characteristics of 4X4 block for CAVLC encoding.
<a href="#"><u>EncodeChromaDcCoeffsCAVLC_H264</u></a>	Calculates characteristics of 2X2 Chroma DC block for CAVLC encoding.

**Table 16-26 H.264 Video Encoder Functions (continued)**

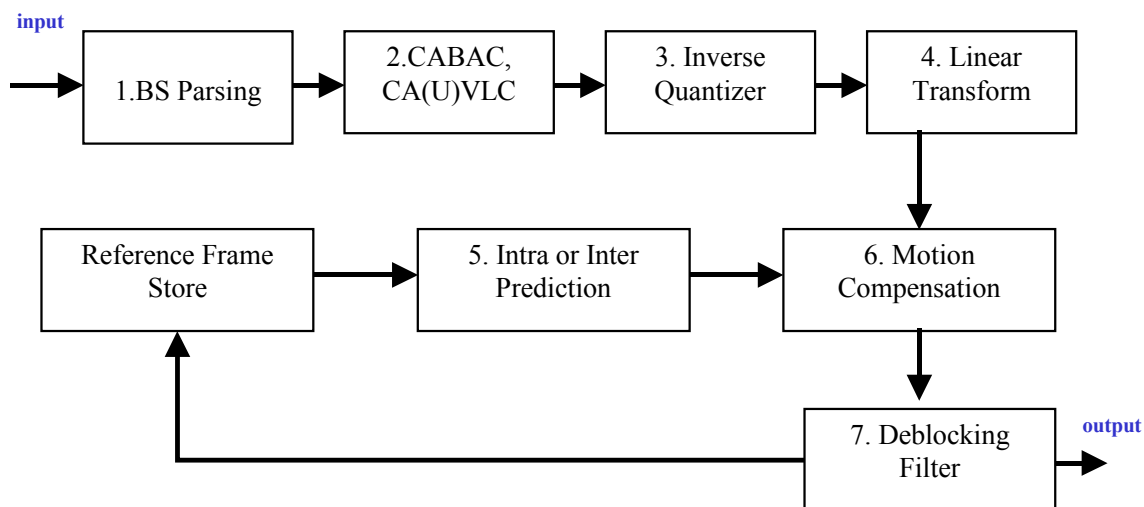
Function Short Name	Description
<b>Inverse Quantization and Transform</b>	
<a href="#"><u>TransformDequantLumaDC H264</u></a>	Performs integer inverse transformation and dequantization for 4x4 luma DC coefficients.
<a href="#"><u>TransformDequantChromaDC H264</u></a>	Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients.
<a href="#"><u>DequantTransformResidualAndAdd H264</u></a>	Performs dequantization, integer inverse transformation, shift for a 4x4 block of residuals with subsequent intra prediction or motion compensation.
<a href="#"><u>QuantLuma8x8Inv H264</u></a>	Performs inverse quantization for 8x8 Luma block coefficients including normalization of the following inverse 8x8 transform.
<a href="#"><u>TransformLuma8x8InvAddPred H264</u></a>	Performs inverse 8x8 transform of 8x8 Luma block coefficients with subsequent intra prediction or motion compensation.
<b>Deblocking</b>	
<a href="#"><u>FilterDeblockingLuma_VerEdge H264</u></a>	Performs deblocking filtering on the vertical edges of the luma 16x16 macroblock.
<a href="#"><u>FilterDeblockingLuma_VerEdge MBAFF H264</u></a>	Performs deblocking filtering on the vertical edges of half of the luma 16x16 macroblock.
<a href="#"><u>FilterDeblockingLuma_HorEdge H264</u></a>	Performs deblocking filtering on the horizontal edges of the luma 16x16 macroblock.
<a href="#"><u>FilterDeblockingChroma_VerEdge H264</u></a>	Performs deblocking filtering on the vertical edges of the chroma 8x8 macroblock.
<a href="#"><u>FilterDeblockingChroma_VerEdge MBAFF H264</u></a>	Performs deblocking filtering on the vertical edges of half of the chroma 8x8 macroblock.
<a href="#"><u>FilterDeblockingChroma_HorEdge H264</u></a>	Performs deblocking filtering on the horizontal edges of the chroma 8x8 macroblock.

## H.264 Decoder Functions

This section describes the functions for H.264 Decoder in accordance with JVT-G050 ([JVTG050]) standard.

**Figure 16-49 H.264 Decoder Structure**

---

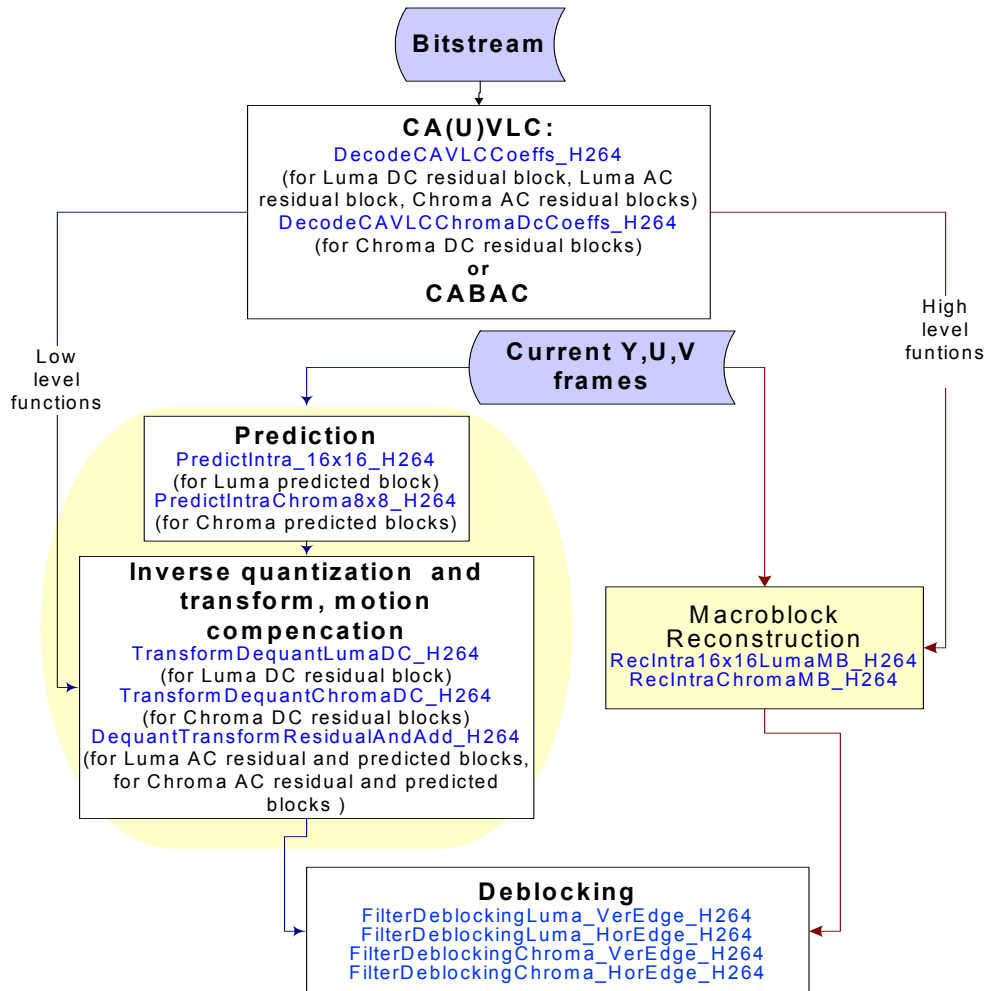


**NOTE.** This Motion Compensation process is performed by general motion compensation functions (See [General Functions](#)), which sum the predictor data and the residual and stores the result into the destination picture.

---

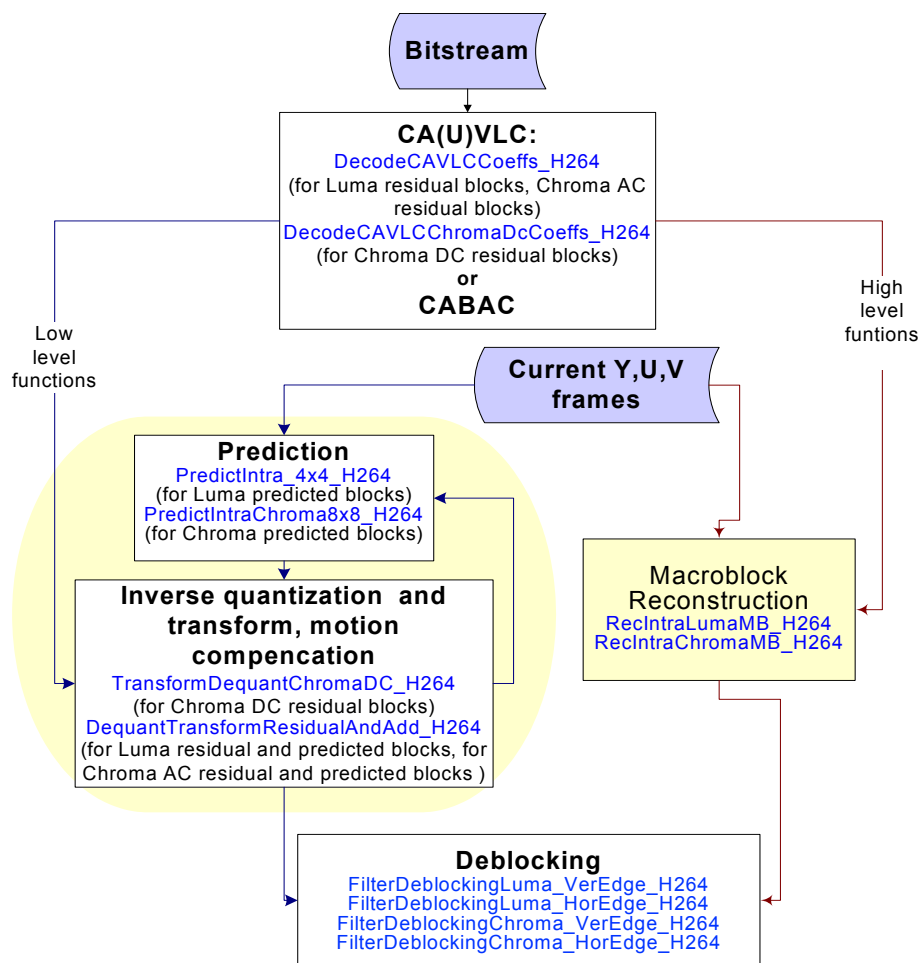
The following schemes show macroblock decoding with the help of ippVC functions. The schemes for Intra16x16, Intra 4x4, and P or B macroblock decoding show two paths of ippVC functions using low and high level functions respectively. The blocks marked with yellow background color are interchangeable.

**Figure 16-50 Intra 16x16 Macroblock Decoding**



The following graph shows decoding process for I macroblock type in the case of Intra\_4x4 prediction mode.

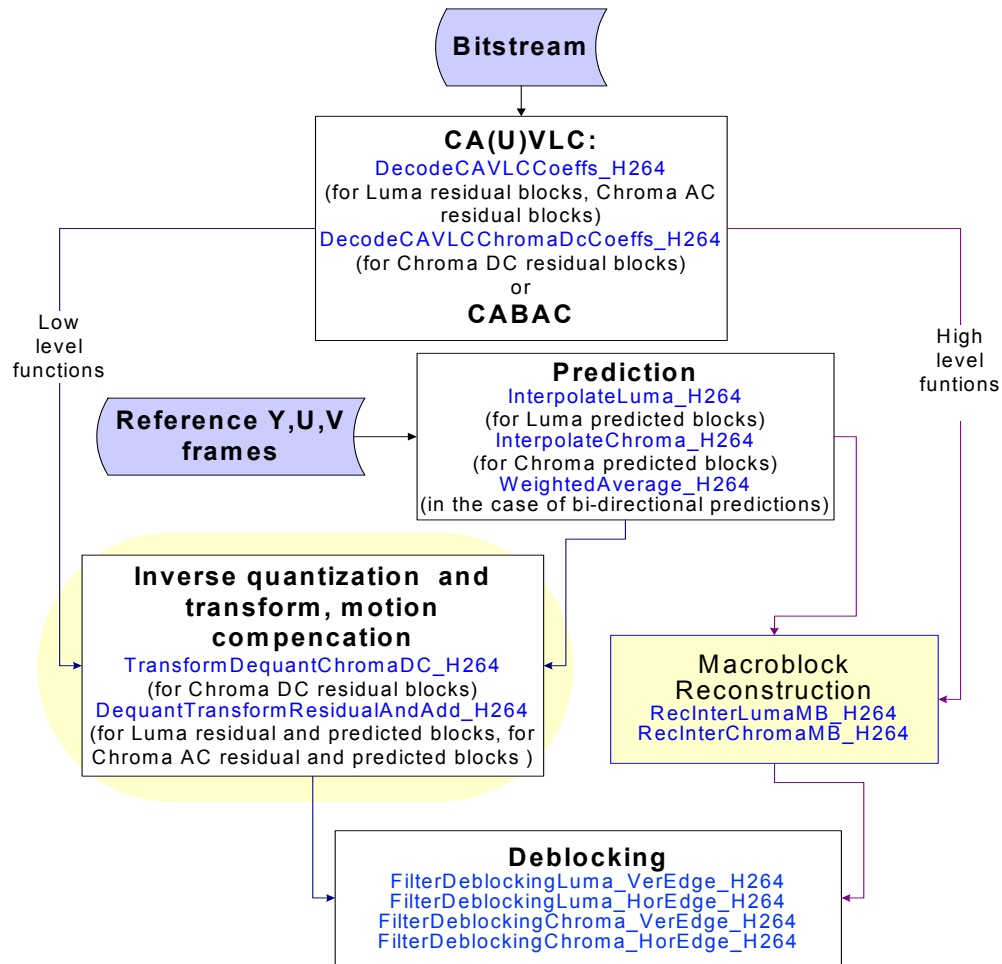
**Figure 16-51 Intra 4x4 Macroblock Decoding**





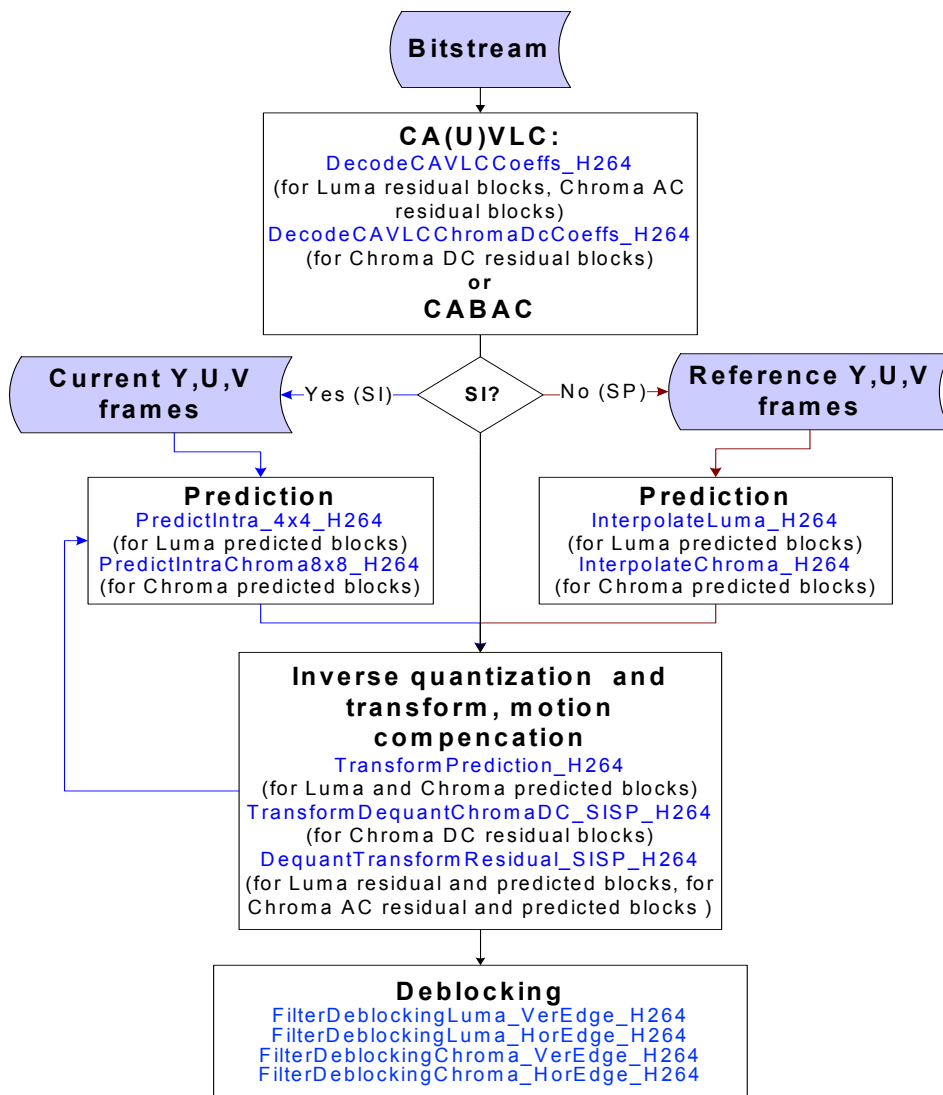
The following graph shows decoding process for P or B macroblock type.

**Figure 16-52 P or B Macroblock Decoding**



The following graph shows decoding process for SI or SP macroblock type. Note that SI is an intra macroblock and SP is a predicted macroblock but their decoding schemes are similar.

### Figure 16-53 SI and SP Macroblocks Decoding



## CAVLC Parsing

### DecodeCAVLCCoeffs\_H264

*Decodes any non-Chroma DC coefficients CAVLC coded.*

#### Syntax

```
IppStatus ippiDecodeCAVLCCoeffs_H264_1u16s(Ipp32u **ppBitStream, Ipp32s
    *pBitOffset, Ipp16s *pNumCoeff, Ipp16s **ppDstCoeffs, const Ipp32u
    uVLCSelect, const Ipp16s uMaxNumCoeff, const Ipp32s **ppTblCoeffToken,
    Ipp32s **ppTblTotalZeros, Ipp32s **ppTblRunBefore, Ipp32s *pScanMatrix);
```

#### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bitstream. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to offset between the bit that <i>ppBitStream</i> points to and the start of the code. The pointer is updated by the function.
<i>pNumCoeff</i>	Pointer to the output number of non-zero coefficients.
<i>ppDstCoeffs</i>	Double pointer to a 4x4 block of coefficients. These coefficients are calculated by the function. The function shifts pointer <i>*ppDstCoeffs</i> on 16.
<i>uVLCSelect</i>	Predictor on number of CoeffToken Table, which is designated in <a href="#">[JVTG050]</a> as <i>nC</i> . It can be calculated in accordance with 9.2.1 of <a href="#">[JVTG050]</a>
<i>uMaxNumCoeff</i>	Maximum coefficients in block (16 for Intra 16x16, 15 for the rest).
<i>ppTblCoeffToken</i>	Double pointer to CoeffToken Tables.
<i>ppTblTotalZeros</i>	Double pointer to TotalZeros Tables.
<i>ppTblRunBefore</i>	Double pointer to RunBefore Tables.
<i>pScanMatrix</i>	Inverse scan matrix for coefficients in block.

## Description

This function is declared in the `ippvc.h` header file. The function `ippiDecodeCAVLCCoeffs_H264_1u16s` decodes any non-Chroma DC (Chroma AC and Luma) coefficients CAVLC-coded in accordance with 9.2 of [\[JVTG050\]](#).

The table `pTblCoeffToken` is an array of size 4. Each element of this array is a pointer to a table that contains codes, number of trailing one transform coefficient and total number of non-zero transform coefficients in accordance with Table 9-5 of [\[JVTG050\]](#).

- `pTblCoeffToken[0]`, where  $0 \leq uVLCSelect < 2$ ;
- `pTblCoeffToken[1]`, where  $2 \leq uVLCSelect < 4$ ;
- `pTblCoeffToken[2]`, where  $4 \leq uVLCSelect < 8$ ;
- `pTblCoeffToken[3]`, where  $uVLCSelect = -1$  (used only for [DecodeCAVLCChromaDcCoeffs\\_H264](#));
- If  $uVLCSelect > 8$ , this function uses its own table in accordance with table 9-5 of [\[JVTG050\]](#),  $nC > 8$ )

Use function [HuffmanRunLevelTableInitAlloc](#) for creation of tables `pTblCoeffToken[i]`.

The table `ppTblTotalZeros` is an array of size 16. Each element of this array (except 0) is a pointer to a table that contains codes and values (`total_zeros` in [\[JVTG050\]](#)) in accordance with tables 9-7 and 9-8 of [\[JVTG050\]](#)[\[JVTG050\]](#).

- `ppTblTotalZeros[0]` is not used.
- `ppTblTotalZeros[i]` contains codes and values that correspond to `TotalCoeff` (in [\[JVTG050\]](#)) =  $i$ ,  $0 < i < 16$ .

The table `ppTblRunBefore` is array of size 8. Each element of this array (except 0) is a pointer to a table that contains codes and values (`run_before` in [\[JVTG050\]](#)) in accordance with table 9-10 of [\[JVTG050\]](#).

- `ppTblRunBefore[0]` is not used.
- `ppTblRunBefore[i]` contains codes and values that correspond with `zerosLeft` (in [\[JVTG050\]](#)) =  $i$ ,  $0 < i < 7$ .
- `ppTblRunBefore[7]` contains codes and values that correspond with `zerosLeft` (in [\[JVTG050\]](#))  $> 6$ .

Use function [HuffmanTableInitAlloc](#) for creation of tables *PpTblTotalZeros[i]*, *PpTblRunBefore[i]*.



**WARNING.** For proper operation of the function, remove `emulation_prevention_three_byte` described in 7.4.1 of [[JVTG050](#)] from the stream that undergoes decoding.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

## DecodeCAVLCChromaDcCoeffs\_H264

*Decodes Chroma DC coefficients CAVLC coded.*

### Syntax

```
IppStatus ippDecodeCAVLCChromaDcCoeffs_H264_1u16s(Ipp32u **ppBitStream, Ipp32s
    *pBitOffset, Ipp16s *pNumCoeff, Ipp16s **ppDstCoeffs, const Ipp32
    *pTblCoeffToken, const Ipp32s **ppTblTotalZerosCR, const Ipp32s
    **ppTblRunBefore);
```

### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bitstream. The pointer is updated by the function.
<i>pBitOffset</i>	Pointer to offset between the bit that <i>ppBitStream</i> points to and the start of the code. The pointer is updated by the function.
<i>pNumCoeff</i>	Pointer to the output number of coefficients.
<i>ppDstCoeffs</i>	Double pointer to a 2x2 block of coefficients. These coefficients are calculated by the function. The function shifts pointer <i>*ppDstCoeffs</i> on 4.
<i>pTblCoeffToken</i>	Pointer to chroma DC CoeffToken Table.
<i>ppTblTotalZerosCR</i>	Double pointer to chroma DC TotalZeros Tables.

*ppTblRunBefore* Double pointer to RunBefore Tables.

### Description

This function is declared in the `ippvc.h` header file. The function `ippiDecodeCAVLCChromaDcCoeffs_H264_1u16s` decodes Chroma DC coefficients CAVLC-coded in accordance with 9.2 of [JVTG050] in the case of  $nC = -1$ .

*ppTblCoeffToken* is equal to *ppTblCoeffToken*[3]. *ppTblCoeffToken* and *ppTblRunBefore* tables are defined in the same way as in [DecodeCAVLCCoeffs\\_H264](#) function.

The table *ppTblTotalZerosCR* is an array of size 4. Each element of this array is a pointer to a table that contains codes and values (*total\_zeros* in [JVTG050]) in accordance with tables 9-9 of [JVTG050].

- *ppTblTotalZerosCR*[0] is not used.
- *ppTblTotalZerosCR*[*i*] contains codes and values that correspond to *TotalCoeff* (in [JVTG050]) = *i*,  $0 < i < 4$ .




---

**WARNING.** For proper operation of the function, remove `emulation_prevention_three_byte` described in 7.4.1 of [JVTG050] from the stream that undergoes decoding.

---

### Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error when at least one input pointer is NULL.

---

## DecodeExpGolombOne\_H264

*Decodes one Exp-Golomb code according to 9.1 H.264 standard.*

---

### Syntax

```
IppStatus ippiDecodeExpGolombOne_H264_1u16s(Ipp32u **ppBitStream, Ipp32s
    *pBitOffset, Ipp16s *pDst, Ipp8u isSigned);
```

### Parameters

<i>ppBitStream</i>	Double pointer to the current position in the bitstream.
<i>pBitOffset</i>	Pointer to the offset between the bit pointed by <i>pBitStream</i> and the start of the code.
<i>pDst</i>	Pointer to the destination result.
<i>isSigned</i>	Flag that indicates if output value has to be decoded as signed.

### Description

The function `ippiDecodeExpGolombOne_H264_1u16s` is declared in the `ippvc.h` header file. The function decodes one Exp-Golomb code in accordance with 9.1 H.264 standard.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error when at least one input pointer is NULL.

### Inverse Quantization and Inverse Transform

---

## TransformDequantLumaDC\_H264

*Performs integer inverse transformation and dequantization for 4x4 luma DC coefficients.*

---

### Syntax

```
IppStatus ippiTransformDequantLumaDC_H264_16s_C1I(Ipp16s* pSrcDst, Ipp32s QP);
```

### Parameters

<i>pSrcDst</i>	Pointer to the initial coefficients and resultant DC (4x4 block).
<i>QP</i>	Quantization parameter.

### Description

The function `ippiTransformDequantLumaDC_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs integer inverse transformation and dequantization for 4x4 luma DC coefficients in accordance with 8.5.6 of [JVTG050].

The argument  $QP$  stands here for a luma quantization parameter, which is designated in JVT-G050 as  $QP_Y$ .

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if $QP$ is less than 1 or greater than 51.

---

## TransformDequantChromaDC\_H264

*Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients.*

---

### Syntax

```
IppStatus ippiTransformDequantChromaDC_H264_16s_C1I(Ipp16s* pSrcDst, Ipp32s
    QP);
```

### Parameters

$pSrcDst$	Pointer to the initial coefficients and resultant chroma DC (2x2 block).
$QP$	Quantization parameter.

### Description

The function `ippiTransformDequantChromaDC_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients in accordance with 8.5.7 of [JVTG050].

The argument  $QP$  stands here for a chroma quantization parameter, which is designated in JVT-G050 as  $QP_C$ .



This function is used in the H.264 encoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if $QP$ is less than 1 or greater than 51.

---

## DequantTransformResidual\_H264

*Performs dequantization, integer inverse transformation, and shift for a 4x4 block of residuals.*

---

### Syntax

```
IppStatus ippiDequantTransformResidual_H264_16s_C1I(Ipp16s* pSrcDst, Ipp32s
    step, Ipp16s* pDC, Ipp32s AC, Ipp32s QP);
```

### Parameters

<code>pSrcDst</code>	Pointer to the initial residual coefficients 4x4 block (AC luma or AC chroma or luma) and resultant transformed 4x4 block - source&destination array of size 16.
<code>step</code>	Source/destination array step in bytes.
<code>pDC</code>	Pointer to the DC coefficient. If the case of luma block (not Intra 16x16 macroblock type), <code>pDC</code> is set to NULL.
<code>AC</code>	Flag that is not equal to zero, if at least one AC coefficient exists, and is equal to zero otherwise. In the case of luma block, which is not Intra 16x16 macroblock type, AC should be: <ul style="list-style-type: none"> <li>= 0, if only <code>pSrcDst[0]</code> is not equal to 0 and other components are equal to 0,</li> <li>= 1, if any <code>pSrcDst[i]</code> is non-zero (<math>i</math> is within range <math>[1, 15]</math>).</li> </ul>
<code>QP</code>	Quantization parameter for luma or for chroma.

## Description

The function `ippiDequantTransformResidual_H264_16s_C1I` is declared in the `ippvc.h` file and performs scaling, inverse transformation, and shift for a residual 4x4 block described in 8.5.8 of [JVTG050].

For transformation of a chroma block this function is called after [TransformDequantChromaDC\\_H264](#). In the case of 16x16 Intra prediction mode, this function should be called after [TransformDequantLumaDC\\_H264](#) for transforming a luma block.

If  $p_{DC}$  is not equal to NULL, the function, before starting the scaling and transformation process, places the DC coefficient specified by  $p_{DC}$  in the top left corner of the source/destination array, which is designated as  $c_{00}$  in JVT-G050. Also during the scaling and transformation process the formula 8-266 ([JVTG050]) is used when calculating  $w_{00}$ .

If the pointer  $p_{DC}$  is NULL, DC coefficient is dequantized like all the other coefficients. During the scaling and transformation process the formula 8-267 ([JVTG050]) is used when calculating  $w_{00}$ .

When calling the function, the argument  $QP$ , which is designated as  $qP$  in JVT-G050, should be calculated from the formulas 8-262 through 8-265 ([JVTG050]).

If  $AC$  is not equal to zero, formulas 8-278 through 8-277 ([JVTG050]) are applied to perform Inverse Transformations.

If  $AC = 0$ , all the coefficients are set equal to DC. The final shift is specified by formula 8-278.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if $QP$ is less than 1 or greater than 51.
<code>ippStsStepErr</code>	Indicates an error condition if $srcStep$ value is less than 8.

---

## DequantTransformResidualAndAdd\_H264

*Performs dequantization, integer inverse transformation, shift for a 4x4 block of residuals with subsequent intra prediction or motion compensation.*

---

### Syntax

```
IppStatus ippiDequantTransformResidualAndAdd_H264_16s_C1I(const Ipp8u* pPred,
    Ipp16s* pSrcDst, Ipp16s* pDC, Ipp8u* pDst, Ipp32s PredStep, Ipp32s DstStep,
    Ipp32s QP, Ipp32s AC);
```

### Parameters

<i>pPred</i>	Pointer to the reference 4x4 block, which is used for intra prediction or motion compensation.
<i>pSrcDst</i>	If not equal to 0, pointer to the initial residual 4x4 block (AC luma or AC chroma or luma) and resultant transformed 4x4 block - array of size 16. If all coefficients of the residual AC block are equal to 0, <i>pSrcDst</i> should be 0 (for chroma block or luma block of 16x16 Intra macroblock type).
<i>pDC</i>	Pointer to the DC coefficient. If luma block is not Intra 16x16 macroblock type, <i>pDC</i> is set to NULL.
<i>pDst</i>	Pointer to the destination 4x4 block.
<i>PredStep</i>	Reference frame step in bytes.
<i>DstStep</i>	Destination frame step in bytes.
<i>QP</i>	Quantization parameter for luma and for chroma.
<i>AC</i>	Flag that is not equal to zero, if at least one AC coefficient exists, and is equal to zero otherwise. In the case of luma block, which is not Intra 16x16 macroblock type, AC should be: <ul style="list-style-type: none"> <li>• = 0, if only <i>pSrcDst</i>[0] is not equal to 0 and other components are equal to 0,</li> <li>• = 1, if any <i>pSrcDst</i>[<i>i</i>] is non-zero (<i>i</i> is within range [1, 15]).</li> </ul>

## Description

The function `ippiDequantTransformResidualAndAdd_H264_16s_C1I` is declared in the `ippvc.h` file and performs scaling, transformation, and shift for a residual 4x4 block described in 8.5.8 of [JVTG050] with subsequent intra prediction or motion compensation (the formula 8-245 of [JVTG050]).

For transformation of a chroma block this function should be called after [TransformDequantChromaDC\\_H264](#). In the case of luma block (16x16 Intra prediction mode), this function should be called after [TransformDequantLumaDC\\_H264](#) for transforming a luma block.

If  $p_{DC}$  is not equal to NULL, the function, before starting the scaling and transformation process, places the DC coefficient specified by  $p_{DC}$  in the top left corner of the initial coefficients block, which is designated as  $c_{00}$  in JVT-G050. Also during the scaling and transformation process the formula 8-266 ([JVTG050]) is used when calculating  $w_{00}$ .

If the pointer  $p_{DC}$  is NULL, DC coefficient is dequantized like all the other coefficients. During the scaling and transformation process the formula 8-267 ([JVTG050]) is used when calculating  $w_{00}$ .

When calling the function, the argument  $Q_P$ , which is designated as  $q_P$  in JVT-G050, should be calculated from the formulas 8-262 through 8-265 ([JVTG050]).

If  $AC$  is not equal to zero, formulas 8-278 through 8-277 ([JVTG050]) are applied to perform Inverse Transformations.

If  $AC = 0$ , all the coefficients are set equal to DC. The final shift is specified by formula 8-278.

This function is used in the H.264 encoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippiStsNoErr</code>	Indicates no error.
<code>ippiStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippiStsOutOfRangeErr</code>	Indicates an error condition if $Q_P$ is less than 1 or greater than 51.

---

## TransformPrediction\_H264

*Performs inverse transform of inter prediction samples for the current macroblock in decoding process for P macroblocks in SP slices or SI macroblocks.*

---

### Syntax

```
IppStatus ippiTransformPrediction_H264_8u16s_C1(Ipp8u *pSrc, Ipp32s step, Ipp16s *pDst);
```

### Parameters

<i>pSrc</i>	Pointer to the array of inter prediction samples for the current macroblock (4x4 block).
<i>step</i>	Step of the <i>pSrc</i> array.
<i>pDst</i>	Destination array of size 16.

### Description

The function `ippiTransformPrediction_H264_8u16s_C1` is declared in the `ippvc.h` file. The function performs inverse transform of inter prediction samples for the current macroblock in decoding process for P macroblocks in SP slices or SI macroblocks (the formula 8-286 of [JVTG050]).

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

---

## DequantTransformResidual\_SISP\_H264

*Performs integer inverse transformation and dequantization of one block in P macroblocks in SP slices or SI macroblocks.*

---

### Syntax

```
IppStatus ippidequantTransformResidual_SISP_H264_16s_C1I(Ipp16s* pSrcDst,  
    Ipp16s* pPredictBlock, Ipp16s* pDC, Ipp32s AC, Ipp32s qp, Ipp32s qs, Ipp32s  
    Switch);
```

### Parameters

<i>pSrcDst</i>	Pointer to array (size 16) of the prediction residual transform coefficient (luma or chroma) - source/destination array.
<i>pPredictBlock</i>	Pointer to array (size 16) of the of inter prediction samples for the current macroblock after inverse transform. They can be calculated using <a href="#">TransformPrediction_H264</a> function.
<i>pDC</i>	Pointer to the DC coefficient. In the case of luma block type <i>pDC</i> must be 0. In the case of chroma block <i>pDC</i> is pointer to array (size 4) of ChromaDC after inverse transform. They can be calculated using <a href="#">TransformDequantChromaDC_SISP_H264</a> function.
<i>AC</i>	Flag that is not equal to zero, if at least one AC coefficient exists, and is equal to zero otherwise.
<i>qp</i>	Quantization parameter <i>qp</i> , which is used in the case of non-switching pictures.
<i>qs</i>	Quantization parameter <i>qs</i> .
<i>Switch</i>	Flag that is not equal to zero, if it switches pictures, and is equal to zero otherwise.

### Description

The function `ippidequantTransformResidual_SISP_H264_16s_C1I` is declared in the `ippvc.h` file. The function performs integer inverse transformation and dequantization of one block in P macroblocks in SP slices or SI macroblocks (8-6 of [JVTG050](#)).

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

---

## TransformDequantChromaDC\_SISP\_H264

*Performs integer inverse transformation and dequantization for 2x2 chroma DC coefficients in P macroblocks in SP slices or SI macroblocks.*

---

### Syntax

```
IppStatus ippTransformDequantChromaDC_SISP_H264_16s_C1I(Ipp16s* pSrcDst,  
Ipp16s* pDCPredict, Ipp32s qp, Ipp32s qs, Ipp32s Switch);
```

### Parameters

<code>pSrcDst</code>	Pointer to array (size 4) of the prediction residual transform chroma DC coefficient - source/destination array.
<code>pDCPredict</code>	Pointer to array (size 4) of the of inter prediction DC samples for the current macroblock after inverse transform. Inverse transform of the inter prediction samples can be calculated using <a href="#">TransformPrediction_H264</a> function for each 4x4 block.
<code>qp</code>	Quantization parameter ( $QP$ . in <a href="#">[JVTG050]</a> ), which is used in the case of non-switching pictures.
<code>qs</code>	Quantization parameter ( $QS$ . in <a href="#">[JVTG050]</a> ).
<code>Switch</code>	Flag that is not equal to zero, if it switches pictures, and is equal to zero otherwise.

## Description

The function `ippiTransformDequantChromaDC_SISP_H264_16s_C1I` is declared in the `ippvc.h` file. The function performs integer inverse transformation and dequantization for 2x2 chroma coefficients in P macroblocks in SP slices or SI macroblocks (8.6.1.2, 8.6.2.2 of [JVTG050]).

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## Intra Prediction

---

## PredictIntra\_4x4\_H264

*Performs intra prediction for a 4x4 luma component.*

---

## Syntax

```
IppStatus ippiPredictIntra_4x4_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s srcdstStep,
      IppIntra4x4PredMode predMode, Ipp32s availability);
```

## Parameters

<code>pSrcDst</code>	Pointer to the source and destination array.
<code>srcdstStep</code>	Step of the source and destination array (Y-frame width in bytes).
<code>predMode</code>	Prediction mode of the <code>Intra_4x4</code> prediction process for luma samples.
<code>availability</code>	Flag that specifies availability of the samples used for prediction.



Description

The function `ippiPredictIntra_4x4_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs intra prediction for a 4x4 luma component in accordance with 8.3.1.2 of [JVTG050] in a given prediction mode *predMode* (See enumeration [IppIntra4x4PredMode](#)).

The flag *availability* is computed as follows:

$$B1*IPP\_LEFT + B2*IPP\_UPPER\_LEFT + B3*IPP\_UPPER + B4*IPP\_UPPER\_RIGHT,$$

where the constants `IPP_LEFT`, `IPP_UPPER_LEFT`, `IPP_UPPER`, `IPP_UPPER_RIGHT` are from [IppLayoutFlag](#) and the variables `B1`, `B2`, `B3`, `B4` may take the values of

- 0 - if the block is unavailable for 4x4 prediction,
- 1 - if the block is available for 4x4 prediction.

Figure 16-54    Layout of Blocks Used for Prediction

2- IPP_UPPER_LEFT	3- IPP_UPPER	4- IPP_UPPER_RIGHT
1- IPP_LEFT	SrcDst	

Return Values

- `ippStsNoErr`                Indicates no error.
- `ippStsNullPtrErr`       Indicates an error condition if at least one of the specified pointers is NULL.
- `ippStsStepErr`           Indicates an error condition if *srcdstStep* value is less than 4.
- `ippStsOutOfRangeErr`   Indicates an error condition if *predMode* value falls outside [0,8].
- `ippStsResFloor`          Indicates a warning condition if *predMode* is not allowed for this block.

## PredictIntra\_16x16\_H264

*Performs intra prediction for a 16x16 luma component.*

---

### Syntax

```
IppStatus ippPredictIntra_16x16_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s
    srcdstStep, IppIntra16x16PredMode predMode, Ipp32s availability);
```

### Parameters

<i>pSrcDst</i>	Pointer to the destination array.
<i>srcdstStep</i>	Step of the destination array in bytes.
<i>predMode</i>	Prediction mode of the Intra_16x16 prediction process for luma samples.
<i>availability</i>	Flag that specifies the availability of the macroblocks used for prediction.

### Description

The function `ippPredictIntra_16x16_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs intra prediction for a 16x16 luma component in accordance with 8.3.2 of [JVTG050] in a given prediction mode *predMode* (See enumeration [IppIntra16x16PredMode](#)).

The flag *availability* is computed as follows:

$$B1 * IPP\_LEFT + B2 * IPP\_UPPER\_LEFT + B3 * IPP\_UPPER,$$

where the constants `IPP_LEFT`, `IPP_UPPER_LEFT`, `IPP_UPPER` are from [IppLayoutFlag](#) and the variables `B1`, `B2`, `B3` may take the values of

0 - if the macroblock is unavailable for 16x16 prediction,

1 - if the macroblock is available for 16x16 prediction.

Figure 16-55    Layout of Blocks Used for Prediction

2- IPP_UPPER_LEFT	3- IPP_UPPER
1- IPP_LEFT	<i>SrcDst</i>

Return Values

- ippStsNoErr

Indicates no error.
- ippStsNullPtrErr

Indicates an error condition if at least one of the specified pointers is NULL.
- ippStsStepErr

Indicates an error condition if *srcdstStep* value is less than 16.
- ippStsOutOfRangeErr

Indicates an error condition if *predMode* value falls outside [0,3].
- ippStsLPCCalcErr

Indicates an error condition if *predMode* is not allowed for this block.

PredictIntraChroma8x8\_H264

*Performs intra prediction for a 8x8 chroma component.*

Syntax

```
IppStatus ippiPredictIntraChroma8x8_H264_8u_C1IR(Ipp8u* pSrcDst, Ipp32s
srcdstStep, IppIntraChroma8x8PredMode predMode, Ipp32s availability);
```

Parameters

- pSrcDst*

Pointer to the 8x8 chroma-block, which is inside U-frame or V-frame and which is under reconstruction (destination array).
- srcdstStep*

Step of the destination array (width of the chroma-frame in bytes).
- predMode*

Prediction mode of the *Intra\_chroma* prediction process for chroma samples.

*availability* Flag that specifies the availability of the macroblocks used for prediction.

## Description

The function `ippiPredictIntraChroma8x8_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs intra prediction for a 8x8 chroma component in accordance with 8.3.3 of [JVTG050] in a given prediction mode *predMode* (See enumeration [IppIntraChromaPredMode\\_H264](#)).

The flag *availability* is computed as follows:

$$B1 * IPP\_LEFT + B2 * IPP\_UPPER\_LEFT + B3 * IPP\_UPPER,$$

where the constants `IPP_LEFT`, `IPP_UPPER_LEFT`, `IPP_UPPER` are from [IppLayoutFlag](#) and the variables `B1`, `B2`, `B3` may take the values of

- 0 - if the macroblock is unavailable for Intra chroma prediction,
- 1 - if the macroblock is available for Intra chroma prediction.

**Figure 16-56 Layout of Blocks Used for Prediction**

---

2- IPP_UPPER_LEFT	3- IPP_UPPER
1- IPP_LEFT	<i>SrcDst</i>

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error condition if <i>predMode</i> value falls outside [0,3].
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcdstStep</i> value is less than 8.
<code>ippStsLPCCalcErr</code>	Indicates an error condition if <i>predMode</i> is not allowed for this block.

---

## Inter Prediction

---

### ExpandPlane\_H264

*Expands plane.*

---

#### Syntax

```
IppStatus ippiExpandPlane_H264_8u_C1R(Ipp8u *StartPtr, Ipp32u uFrameWidth,  
    Ipp32u uFrameHeight, Ipp32u uPitch, Ipp32u uPels, IppvcFrameFieldFlag  
    uFrameFieldFlag);
```

#### Parameters

<i>StartPtr</i>	Pointer to the frame source data.
<i>uFrameWidth</i>	Frame width in pixels.
<i>uFrameHeight</i>	Frame height in pixels.
<i>uPitch</i>	Frame width in bytes.
<i>uPels</i>	Number of pixels filled to the right/left and up/down in process of expansion.
<i>uFrameFieldFlag</i>	Flag that defines expansion method for various image types: frame, top field, bottom field. See the corresponding enumeration <a href="#">IPPVC_FRAME_FIELD_FLAG</a> .

#### Description

The function `ippiExpandPlane_H264_8u_C1R` is declared in the `ippvc.h` file. This function expands the plane by adding pixels to the top and bottom rows and to the left and right columns of the frame.

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

#### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if <i>StartPtr</i> is NULL.

---

## InterpolateLuma\_H264

*Performs interpolation for motion estimation of the luma component.*

---

### Syntax

```
IppStatus ippiInterpolateLuma_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep,  
    Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, IppiSize roiSize);
```

### Parameters

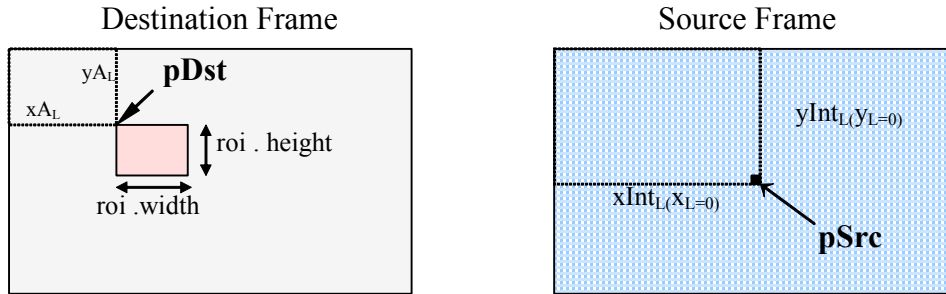
<i>pSrc</i>	Pointer to the source.
<i>srcStep</i>	Step of the pointer <i>pSrc</i> (source array) in bytes.
<i>pDst</i>	Pointer to the destination.
<i>dstStep</i>	Step of the pointer <i>pDst</i> (destination array) in bytes.
<i>dx</i> , <i>dy</i>	Fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3).
<i>roiSize</i>	Flag that specifies the dimensions of the region of interest (could be 16, 8 or 4 in each dimension). See structure <a href="#">IppiSize</a> .

### Description

The function `ippiInterpolateLuma_H264_8u_C1R` is declared in the `ippvc.h` file. This function performs interpolation (convolution with 6x6 kernel) for motion estimation of the luma component in accordance with 8.4.2.2.1 of [\[JVTG050\]](#).

The function uses only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

**Figure 16-57 Interpolation for Motion Estimation of Luma Component**



**NOTE.** Make sure all samples that are used for interpolation are within the boundaries of the source frame. To solve the problem, enlarge the source frame by using boundary samples or use [InterpolateLumaTop H264](#) or [InterpolateLumaBottom H264](#) functions.

This function is used in the H.264 decoder and encoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is less than 16.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>dx</i> or <i>dy</i> values fall outside [0,3].
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> take values other than {16, 8, 4}.

---

## InterpolateLumaTop\_H264

*Performs interpolation for motion estimation of the luma component at the frame top boundary.*

---

### Syntax

```
IppStatus ippiInterpolateLumaTop_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep,  
      Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s outPixels,  
      IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source.
<i>srcStep</i>	Step of the pointer <i>pSrc</i> (source array) in bytes.
<i>pDst</i>	Pointer to the destination.
<i>dstStep</i>	Step of the pointer <i>pDst</i> (destination array) in bytes.
<i>dx</i> , <i>dy</i>	Fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3).
<i>outPixels</i>	Number of pixels by which the data specified by <i>pSrc</i> reaches over the frame top boundary.
<i>roiSize</i>	Flag that specifies the dimensions of the region of interest (could be 16, 8 or 4 in each dimension). See structure <a href="#">IppiSize</a> .

### Description

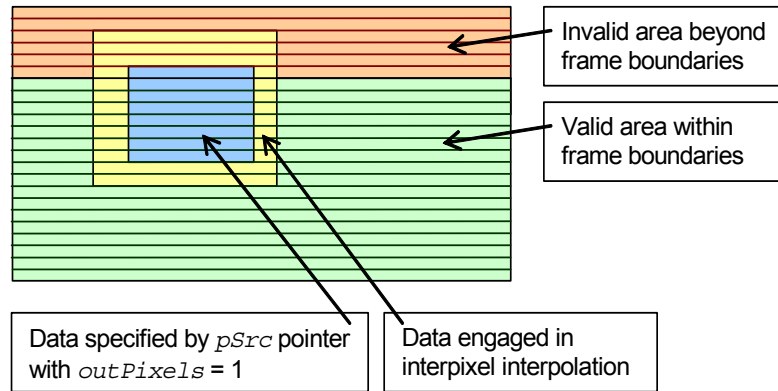
The function `ippiInterpolateLumaTop_H264_8u_C1R` is declared in the `ippvc.h` file. This function performs interpolation (convolution with 6x6 kernel) for motion estimation of the luma component near the top boundary of the frame in accordance with 8.4.2.2.1 of [\[JVTG050\]](#). Instead of data lines located in the invalid area, the closest line in the valid area is used.

The function uses only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

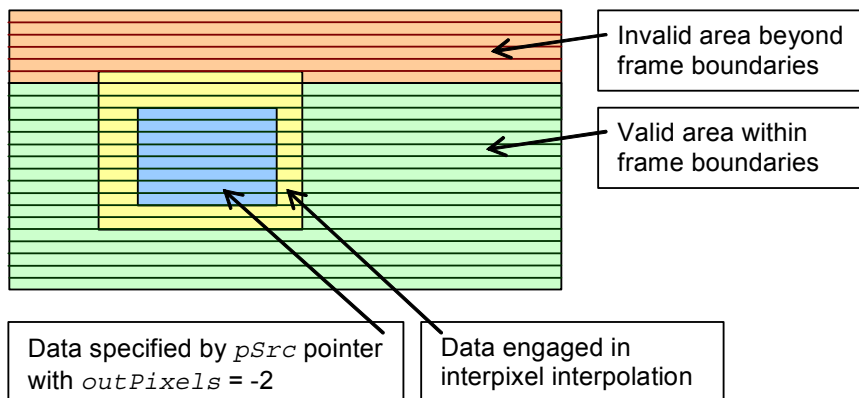
[Figure 16-58](#) shows the data specified by *pSrc* and the data actually used in interpixel interpolation at the picture top boundary with a positive *outPixels* value. [Figure 16-59](#) shows the data with a negative *outPixels* value.



**Figure 16-58 Luma Interpolation at Top Boundary with Positive *outPixels* Value**



**Figure 16-59 Luma Interpolation at Top Boundary with Negative *outPixels* Value**



## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is less than 16.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>dx</i> or <i>dy</i> values fall outside [0,3].
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> take values other than {16, 8, 4}.

---

## InterpolateLumaBottom\_H264

*Performs interpolation for motion estimation of the luma component at the frame bottom boundary.*

---

### Syntax

```
IppStatus ippInterpolateLumaBottom_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s
srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s
outPixels, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source.
<i>srcStep</i>	Step of the pointer <i>pSrc</i> (source array) in bytes.
<i>pDst</i>	Pointer to the destination.
<i>dstStep</i>	Step of the pointer <i>pDst</i> (destination array) in bytes.
<i>dx</i> , <i>dy</i>	Fractional parts of the motion vector in 1/4 pel units (0, 1, 2, or 3).
<i>outPixels</i>	Number of pixels by which the data specified by <i>pSrc</i> reaches over the frame bottom boundary.
<i>roiSize</i>	Flag that specifies the dimensions of the region of interest (could be 16, 8 or 4 in each dimension). See structure <a href="#">IppiSize</a> .

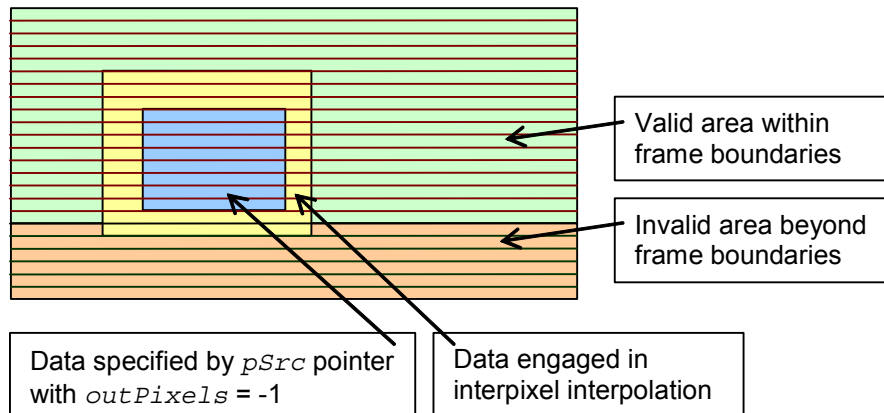
## Description

The function `ippiInterpolateLumaBottom_H264_8u_C1R` is declared in the `ippvc.h` file. This function performs interpolation (convolution with 6x6 kernel) for motion estimation of the luma component near the bottom boundary of the frame in accordance with 8.4.2.2.1 of [JVTG050]. Instead of data lines located in the invalid area, the closest line in the valid area is used.

The function uses only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by `pSrc`) in the source frame.

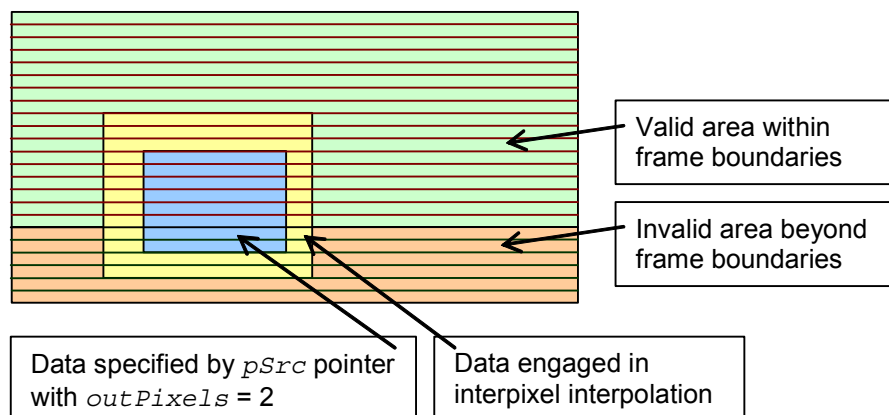
[Figure 16-60](#) shows the data specified by `pSrc` and the data actually used in interpixel interpolation at the picture top boundary with a positive `outPixels` value. [Figure 16-61](#) shows the data with a negative `outPixels` value.

**Figure 16-60 Luma Interpolation at Bottom Boundary with Positive `outPixels` Value**



**Figure 16-61 Luma Interpolation at Bottom Boundary with Negative *outPixels* Value**

---



### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is less than 16.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>dx</i> or <i>dy</i> values fall outside [0,3].
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> take values other than {16, 8, 4}.

---

## InterpolateChroma\_H264

*Performs interpolation for motion estimation of the chroma component.*

---

### Syntax

```
IppStatus ippiInterpolateChroma_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s srcStep,  
      Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, IppiSize roiSize);
```

### Parameters

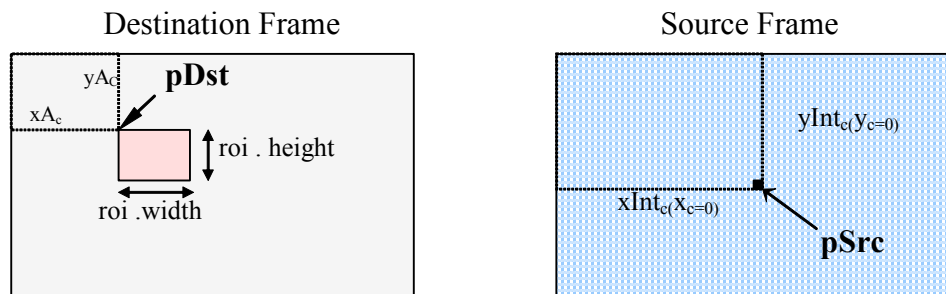
<i>pSrc</i>	Pointer to the source.
<i>srcStep</i>	Step of the pointer <i>pSrc</i> in bytes.
<i>pDst</i>	Pointer to the destination.
<i>dstStep</i>	Step of the pointer <i>pDst</i> in bytes.
<i>dx</i> , <i>dy</i>	Fractional parts of the motion vector in 1/8 pel units (0, 1, ..., 7).
<i>roiSize</i>	Flag that specifies the dimensions of the region of interest (could be 8, 4 or 2 in each dimension). See structure <a href="#">IppiSize</a> .

### Description

The function `ippiInterpolateChroma_H264_8u_C1R` is declared in the `ippvc.h` file. This function performs interpolation for motion estimation of the chroma component in accordance with 8.4.2.2.2 of [JVTG050].

The function uses only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

**Figure 16-62 Interpolation for Motion Estimation of Chroma Component**



**NOTE.** Make sure all samples that are used for interpolation are within the boundaries of the source frame. To solve the problem, enlarge the source frame by using boundary samples or use [InterpolateChromaTop\\_H264](#) or [InterpolateChromaBottom\\_H264](#) functions.

This function is used in the H.264 decoder and encoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is less than 8.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>dx</i> or <i>dy</i> values fall outside [0,7].
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> take values other than {8, 4, 2}.

---

## InterpolateChromaTop\_H264

*Performs interpolation for motion estimation of the chroma component at the frame top boundary.*

---

### Syntax

```
IppStatus ippiInterpolateChromaTop_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s
    srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s
    outPixels, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source.
<i>srcStep</i>	Step of the pointer <i>pSrc</i> (source array) in bytes.
<i>pDst</i>	Pointer to the destination.
<i>dstStep</i>	Step of the pointer <i>pDst</i> (destination array) in bytes.
<i>dx, dy</i>	Fractional parts of the motion vector in 1/8 pel units (0, 1, ..., 7).
<i>outPixels</i>	Number of pixels by which the data specified by <i>pSrc</i> reaches over the frame top boundary.
<i>roiSize</i>	Flag that specifies the dimensions of the region of interest (could be 8, 4 or 2 in each dimension). See structure <a href="#">IppiSize</a> .

### Description

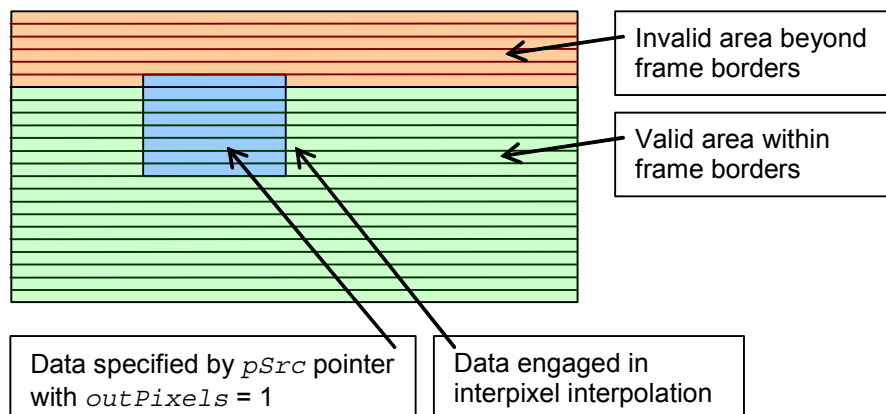
The function `ippiInterpolateChromaTop_H264_8u_C1R` is declared in the `ippvc.h` file. This function performs interpolation for motion estimation of the chroma component near the top boundary of the frame in accordance with 8.4.2.2.2 of [JVTG050]. Instead of data lines located in the invalid area, the closest line in the valid area is used.

The function uses only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

[Figure 16-63](#) shows the data specified by *pSrc* in interpixel interpolation at the picture top boundary.

**Figure 16-63 Chroma Interpolation at Top Boundary**

---



### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is less than 16.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>dx</i> or <i>dy</i> values fall outside [0,3].
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> take values other than {16, 8, 4}.



---

## InterpolateChromaBottom\_H264

*Performs interpolation for motion estimation of the chroma component at the frame bottom boundary.*

---

### Syntax

```
IppStatus ippiInterpolateChromaBottom_H264_8u_C1R(const Ipp8u* pSrc, Ipp32s
    srcStep, Ipp8u* pDst, Ipp32s dstStep, Ipp32s dx, Ipp32s dy, Ipp32s
    outPixels, IppiSize roiSize);
```

### Parameters

<i>pSrc</i>	Pointer to the source.
<i>srcStep</i>	Step of the pointer <i>pSrc</i> (source array) in bytes.
<i>pDst</i>	Pointer to the destination.
<i>dstStep</i>	Step of the pointer <i>pDst</i> (destination array) in bytes.
<i>dx</i> , <i>dy</i>	Fractional parts of the motion vector in 1/8 pel units (0, 1, ..., 7).
<i>outPixels</i>	Number of pixels by which the data specified by <i>pSrc</i> reaches over the frame bottom boundary.
<i>roiSize</i>	Flag that specifies the dimensions of the region of interest (could be 8, 4 or 2 in each dimension). See structure <a href="#">IppiSize</a> .

### Description

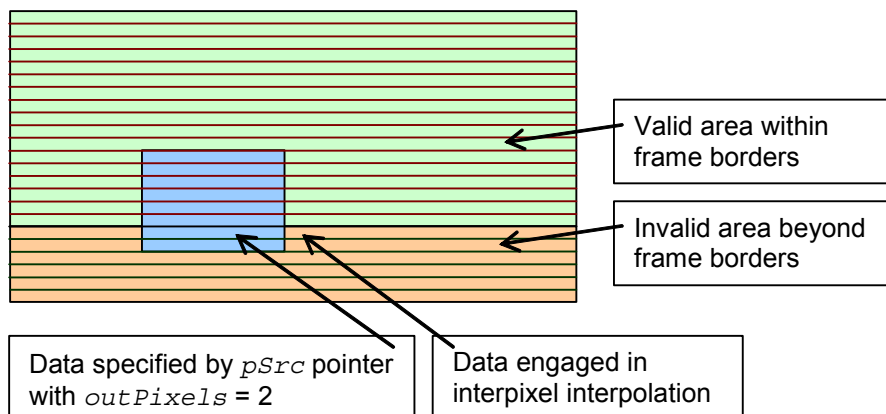
The function `ippiInterpolateChromaBottom_H264_8u_C1R` is declared in the `ippvc.h` file. This function performs interpolation for motion estimation of the chroma component near the bottom boundary of the frame in accordance with 8.4.2.2.2 of [[JVTG050](#)]. Instead of data lines located in the invalid area, the closest line in the valid area is used.

The function uses only the fractional part of the motion vector. The integer part is used when finding the position (pointed to by *pSrc*) in the source frame.

[Figure 16-64](#) shows the data specified by *pSrc* in interpixel interpolation at the picture top boundary.

**Figure 16-64 Chroma Interpolation at Bottom Boundary**

---



### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value is less than 16.
<code>ippStsBadArgErr</code>	Indicates an error condition if <i>dx</i> or <i>dy</i> values fall outside [0,3].
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> take values other than {16, 8, 4}.

---

## InterpolateBlock\_H264

*Calculates average value for each source pair values in block.*

---

### Syntax

```
IppStatus ippiInterpolateBlock_H264_8u_P2P1R(Ipp8u *pSrc1, Ipp8u *pSrc2, Ipp8u
    *pDst, Ipp32u width, Ipp32u height, Ipp32u pitch);
```

### Parameters

<i>pSrc1</i>	Pointer to the first input block.
<i>pSrc2</i>	Pointer to the second input block.
<i>pDst</i>	Pointer to the first output block.
<i>width</i>	Number of values in the block line.
<i>height</i>	Number of lines in the block.
<i>pitch</i>	Memory pitch.

### Description

The function `ippiInterpolateBlock_H264_8u_P2P1R` is declared in the `ippvc.h` file. This function calculates the average value for each corresponding source pair values in a block with height lines and width values per line.

$$pDst[i] = \frac{pSrc1[i] + pSrc2[i] + 1}{2}.$$

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

---

## WeightedAverage\_H264

*Averages two blocks with weights.*

---

### Syntax

```
IppStatus ippiWeightedAverage_H264_8u_C1IR(const Ipp8u* pSrc1, Ipp8u*  
    pSrc2Dst, Ipp32s srcDstStep, Ipp32s weight1, Ipp32s weight2, Ipp32s shift,  
    Ipp32s offset, IppiSize roiSize);
```

### Parameters

<i>pSrc1</i>	Pointer to the first source (output of preceding functions).
<i>pSrc2Dst</i>	Pointer to the second source and result.
<i>srcDstStep</i>	Step of the pointers <i>pSrc1</i> and <i>pSrc2Dst</i> in bytes.
<i>weight1</i> , <i>weight2</i>	Weights.
<i>shift</i>	Shift.
<i>offset</i>	Offset.
<i>roiSize</i>	Flag that specifies the region of interest (could be 16, 8, 4 or 2 in each dimension).

### Description

The function `ippiWeightedAverage_H264_8u_C1IR` is declared in the `ippvc.h` file. This function averages two blocks with weights (for weighted bi-directional predictions) as specified in 8.4.2.3.2 of [JVTG050] when `predFlagL0` and `predFlagL1` are equal to 1.

The function uses the following formula:

$$Source2[x, y] =$$
$$Clip1((weight1 * Source1[x, y] + weight2 * Source2[x, y] + 1 \ll (shift - 1)) \gg shift + offset)$$

where  $x \in [0, roi.width - 1]$ ,  $y \in [0, roi.height - 1]$ ,

$$Clip1(z) = \begin{cases} 0 & ; z < 0 \\ 255 & ; z > 255 \\ z & ; otherwise \end{cases}$$

The above formula corresponds to formula 8-220 of JVT-G050. See [Table 16-27](#) for matching the function arguments and the standard variables.

**Table 16-27     `WeightedAverage_H264` Arguments vs JTV-G050 Variables**

<code>ippiWeightedAverage</code>	8.4.2.3.2 JVT-G050
<code>weight1</code>	$w_0$
<code>weight2</code>	$w_1$
<code>shift</code>	$\log WD + 1$
<code>offset</code>	$(o_0 + o_1 + 1) \gg 1$
<code>roi.width</code>	<code>partWidth</code>
<code>roi.height</code>	<code>partHeight</code>

Values of  $w_0$ ,  $w_1$ ,  $\log WD$ ,  $o_0$ ,  $o_1$  are calculated from formulas 8-221 through 8-244.

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

**Return Values**

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcDstStep</code> value is less than <code>roi.width</code> .

---

## UniDirWeightBlock\_H264

*Weights source block.*

---

**Syntax**

```
IppStatus ippiUniDirWeightBlock_H264_8u_C1IR(Ipp8u *pSrcDst, Ipp32u srcDstStep,
Ipp32u uLog2Wd, Ipp32s iWeight, Ipp32s iOffset, IppiSize roi);
```

**Parameters**

<code>pSrcDst</code>	Pointer to the source and the result.
----------------------	---------------------------------------

<i>srcDstStep</i>	Step of the pointer in bytes.
<i>uLog2wd</i>	$\log_2$ weight denominator.
<i>iWeight</i>	Weight.
<i>iOffset</i>	Offset.
<i>roi</i>	Flag that specifies the region of interest (could be 16, 8, 4 or 2 in each dimension).

## Description

The function `ippiUniDirWeightBlock_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs weighted prediction as specified in 8.4.2.3.2 of [ITUH264] when `predFlagL0` is equal to 1 and `predFlagL1` are equal to 0 or when `predFlagL0` is equal to 0 and `predFlagL1` are equal to 1.

The function uses the following formula:

```
if (logWD ≥ 1)
```

```
    pSrcDst[x,y]= Clip1c(((pSrcDst[x,y] · iWeight + 2uLog2wd-1) » uLog2wd) + iOffset)
```

```
else
```

```
    pSrcDst[x,y]= Clip1c((pSrcDst[x,y] · iWeight) + iOffset)
```

where  $x \in [0, roi.width - 1]$ ,  $y \in [0, roi.height - 1]$ ,

$$\text{Clip1}(z) = \begin{cases} 0 & ; z < 0 \\ 255 & ; z > 255 \\ z & ; \text{otherwise} \end{cases}$$

The above formula corresponds to formula 8-270 and 8-271 of ITUH264.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcDstStep</i> value is less than <i>roi.width</i> .

`ippStsSizeErr` Indicates an error condition if `roi.width` or `roi.height` value is not equal to 2 or 4 or 8 or 16.

---

## BiDirWeightBlock\_H264

*Averages two blocks with two weights and two offsets.*

---

### Syntax

```
IppStatus ippBiDirWeightBlock_H264_8u_P2P1R(Ipp8u *pSrc1, Ipp8u *pSrc2, Ipp8u
    *pDst, Ipp32u srcStep, Ipp32u dstStep, Ipp32u ulog2wd, Ipp32s iWeight1,
    Ipp32s iOffset1, Ipp32s iWeight2, Ipp32s iOffset2, IppiSize roi);
```

### Parameters

<code>pSrc1</code>	Pointer to the first source.
<code>pSrc2</code>	Pointer to the second source.
<code>pDst</code>	Pointer to the result.
<code>srcStep</code>	Step of <code>pSrc1</code> and <code>pSrc2</code> pointers in bytes.
<code>dstStep</code>	Step of <code>pDst</code> pointer in bytes.
<code>ulog2wd</code>	$\log_2$ weight denominator.
<code>iWeight1</code> , <code>iWeight2</code>	Weights.
<code>iOffset1</code> , <code>iOffset2</code>	Offsets.
<code>roi</code>	Flag that specifies the region of interest (could be 16, 8, 4 or 2 in each dimension).

### Description

The function `ippBiDirWeightBlock_H264_8u_P2P1R` is declared in the `ippvc.h` file. This function performs weighted prediction as specified in 8.4.2.3.2 of [ITUH264] when both `predFlagL0` and `predFlagL1` are equal to 1.

The function uses the following formula:

$$pDst[x,y] = \text{Clip1c}(((pSrc1[x,y] \cdot iWeight1 + pSrc2[x,y] \cdot iWeight2 + 2^{ulog2wd-1}) \gg (ulog2wd + 1) + ((iOffset1 + iOffset2 + 1) \gg 2))$$

where  $x \in [0, roi.width - 1]$ ,  $y \in [0, roi.height - 1]$ ,

$$\text{Clip1}(z) = \begin{cases} 0 & ; z < 0 \\ 255 & ; z > 255 \\ z & ; \text{otherwise} \end{cases}$$

The above formula corresponds to formula 8-272 of ITUH264.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <code>srcStep</code> value or <code>dstStep</code> value is less than <code>roi.width</code> .
<code>ippStsSizeErr</code>	Indicates an error condition if <code>roi.width</code> or <code>roi.height</code> value is not equal to 2 or 4 or 8 or 16.

---

## BiDirWeightBlockImplicit\_H264

*Averages two blocks with weights if `uLog2wd` is equal to 5.*

---

### Syntax

```
IppStatus ippBiDirWeightBlockImplicit_H264_8u_P2P1R(Ipp8u *pSrc1, Ipp8u
    *pSrc2, Ipp8u *pDst, Ipp32u srcStep, Ipp32u dstStep, Ipp32s iWeight1, Ipp32s
    iWeight2, IppiSize roi);
```

### Parameters

<code>pSrc1</code>	Pointer to the first source.
<code>pSrc2</code>	Pointer to the second source.
<code>pDst</code>	Pointer to the result.
<code>srcStep</code>	Step of <code>pSrc1</code> and <code>pSrc2</code> pointers in bytes.
<code>dstStep</code>	Step of <code>pDst</code> pointer in bytes.



*iWeight1*, *iWeight2* Weights.

*roi* Flag that specifies the region of interest (could be 16, 8, 4 or 2 in each dimension).

## Description

The function `ippiBiDirWeightBlockImplicit_H264_8u_P2P1R` is declared in the `ippvc.h` file. This function performs implicit mode weighted prediction as specified in 8.4.2.3.2 of [ITUH264].

This function uses formula 8-272 of [ITUH264], but according to 8-273, 8-274, and 8-275 of ITUH264 the  $\log_2$  weight denominator *uLog2wd* is equal to 5, and both *iOffset1*, *iOffset2* offsets are equal to zero. So the function uses the following formula:

$$pDst[x,y] = \text{Clip1}((pSrc1[x,y] \cdot iWeight1 + pSrc2[x,y] \cdot iWeight2 + 32) \gg 6)$$

where  $x \in [0, roi.width - 1]$ ,  $y \in [0, roi.height - 1]$ ,

$$\text{Clip1}(z) = \begin{cases} 0 & ; z < 0 \\ 255 & ; z > 255 \\ z & ; \text{otherwise} \end{cases}$$

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsStepErr</code>	Indicates an error condition if <i>srcStep</i> value or <i>dstStep</i> value is less than <i>roi.width</i> .
<code>ippStsSizeErr</code>	Indicates an error condition if <i>roi.width</i> or <i>roi.height</i> value is not equal to 2 or 4 or 8 or 16.

## Macroblock Reconstruction

---

### ReconstructChromaInterMB\_H264

*Reconstructs Inter Chroma macroblock.*

---

#### Syntax

```
IppStatus ippiReconstructChromaInterMB_H264_16s8u_P2R(Ipp16s **ppSrcCoeff,  
    Ipp8u *pSrcDstUPlane, Ipp8u *pSrcDstVPlane, const Ipp32u srcDstStep, const  
    Ipp32u cbp4x4, const Ipp32u ChromaQP);
```

#### Parameters

<i>ppSrcCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC <i>U</i> -block, 2x2 DC <i>V</i> -block, 4x4 AC <i>U</i> -blocks, 4x4 AC <i>V</i> -blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to macroblock that is reconstructed in current <i>U</i> plane. This macroblock must contain inter prediction samples.
<i>pSrcDstVPlane</i>	Pointer to macroblock that is reconstructed in current <i>V</i> plane. This macroblock must contain inter prediction samples.
<i>srcDstStep</i>	Plane step.
<i>cbp4x4</i>	<p>Coded block pattern.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)) is not equal to 0, 2x2 DC <i>U</i>-block is not zero-filled and exists in <i>ppSrcCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1)) is not equal to 0, 2x2 DC <i>U</i>-block is not zero-filled and exists in <i>ppSrcCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+<i>i</i>)) is not equal to 0, (<math>0 \leq i &lt; 4</math>), <i>i</i>-th 4x4 AC <i>U</i>-block is not zero-filled and exists in <i>ppSrcCoeff</i>.</p>

If `cbp4x4` &  $(1 < (IPPVC\_CBP\_1ST\_CHROMA\_AC\_BITPOS + i + 4))$  is not equal to 0,  $(0 \leq i < 4)$ ,  $i$ -th 4x4 AC  $U$ -block is not zero-filled and exists in `ppSrcCoeff`.

*ChromaQP*

Chroma quantizer ( $QP_C$  in [JVTG050]). It must be within the range  $[0; 39]$ .

## Description

The function `ippiReconstructChromaInterMB_H264_16s8u_P2R` is declared in the `ippvc.h` file. This function reconstructs Inter Chroma macroblock (8x8  $U$  macroblock and 8x8  $V$  macroblock):

1. Performs integer inverse transformation and dequantization for 2x2  $U$  DC coefficients and for 2x2  $V$  DC coefficients in accordance with 8.5.7 of [JVTG050] in the same way as `TransformDequantChromaDC_H264` function does.
2. Performs scaling, integer inverse transformation, and shift for 4x4 AC  $U$ -blocks and shift for 4x4 AC  $V$ -blocks in accordance with 8.5.8 of [JVTG050] in the same way as `DequantTransformResidual_H264` function does. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7 of [JVTG050].
3. Performs adding of 8x8 inter prediction blocks and 8x8 residual blocks in accordance with 8-247 of [JVTG050].

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>ChromaQP</i> is less than 0 or greater than 39.

## ReconstructChromaIntraHalvesMB\_H264

*Reconstructs two halves of Intra Chroma macroblock.*

---

### Syntax

```
IppStatus ippiReconstructChromaIntraHalvesMB_H264_16s8u_P2R(Ipp16s  
    **ppSrcCoeff, Ipp8u *pSrcDstUPlane, Ipp8u *pSrcDstVPlane, Ipp32u  
    srcDstUVStep, IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u cbp4x4,  
    Ipp32u ChromaQP, Ipp8u edgeTypeTop, Ipp8u edgeTypeBottom);
```

### Parameters

<i>ppSrcCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC <i>U</i> -block, 2x2 DC <i>V</i> -block, 4x4 AC <i>U</i> -blocks, 4x4 AC <i>V</i> -blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to current <i>U</i> plane that is reconstructed.
<i>pSrcDstVPlane</i>	Pointer to current <i>V</i> plane that is reconstructed.
<i>srcDstUVStep</i>	Plane step.
<i>intraChromaMode</i>	Prediction mode of the Intra_chroma prediction process for chroma samples. It is defined in the same way as in <a href="#">PredictIntraChroma8x8_H264</a> function.
<i>cbp4x4</i>	<p>Coded block pattern.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)) is not equal to 0, 2x2 DC <i>U</i>-block is not zero-filled and exists in <i>ppSrcCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1)) is not equal to 0, 2x2 DC <i>U</i>-block is not zero-filled and exists in <i>ppSrcCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+<i>i</i>)) is not equal to 0, (0 ≤ <i>i</i> &lt; 4), <i>i</i>-th 4x4 AC <i>U</i>-block is not zero-filled and exists in <i>ppSrcCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+<i>i</i>+4)) is not equal to 0, (0 ≤ <i>i</i> &lt; 4), <i>i</i>-th 4x4 AC <i>U</i>-block is not zero-filled and exists in <i>ppSrcCoeff</i>.</p>

<i>ChromaQP</i>	Chroma quantizer ( $QP_C$ in <a href="#">[JVTG050]</a> ). It must be within the range [0;39].
<i>edgeTop,</i> <i>edgeTypeBottom</i>	Flags that specify the availability of the macroblocks used for prediction. Upper and lower halves of the macroblock may have different prediction vectors, so use two flags: one for the upper half of the macroblock, the other - for the lower half.

## Description

The function `ippiReconstructChromaIntraHalvesMB_H264_16s8u_P2R` is declared in the `ippvc.h` file. This function reconstructs Intra Chroma macroblock (8x8 *U* macroblock and 8x8 *V* macroblock), divided into upper and lower halves, 16x8 each, differing only in prediction, that is, having independent prediction vectors. Otherwise their performance is the same:

1. Performs integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.7 of [\[JVTG050\]](#) in the same way as [TransformDequantChromaDC\\_H264](#) function does.
2. Performs scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.8 of [\[JVTG050\]](#) in the same way as [DequantTransformResidual\\_H264](#) function does. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7 of [\[JVTG050\]](#).
3. Performs intra prediction for an 8x8 *U*-component and for an 8x8 *V*-component in accordance with 8.3.2 of [\[JVTG050\]](#) in the same way as [PredictIntraChroma8x8\\_H264](#) function does.
4. Performs adding of 8x8 intra prediction blocks and 8x8 residual blocks in accordance with 8-247 of [\[JVTG050\]](#).

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>ChromaQP</i> is less than 0 or greater than 39.

## ReconstructChromaIntraMB\_H264

*Reconstructs Intra Chroma macroblock.*

---

### Syntax

```
IppStatus ippiReconstructChromaIntraMB_H264_16s8u_P2R(Ipp16s **ppSrcCoeff,  
Ipp8u *pSrcDstUPlane, Ipp8u *pSrcDstVPlane, const Ipp32u srcDstUVStep, const  
IppIntraChromaPredMode_H264 intraChromaMode, const Ipp32u cbp4x4, const  
Ipp32u ChromaQP, const Ipp8u edgeType);
```

### Parameters

<i>ppSrcCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC <i>U</i> -block, 2x2 DC <i>V</i> -block, 4x4 AC <i>U</i> -blocks, 4x4 AC <i>V</i> -blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to current <i>U</i> plane that is reconstructed.
<i>pSrcDstVPlane</i>	Pointer to current <i>V</i> plane that is reconstructed.
<i>srcDstUVStep</i>	Plane step.
<i>intraChromaMode</i>	Prediction mode of the Intra_chroma prediction process for chroma samples. It is defined in the same way as in <a href="#">PredictIntraChroma8x8_H264</a> function.
<i>cbp4x4</i>	<p>Coded block pattern.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)) is not equal to 0, 2x2 DC <i>U</i>-block is not zero-filled and exists in <i>ppSrcCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1)) is not equal to 0, 2x2 DC <i>U</i>-block is not zero-filled and exists in <i>ppSrcCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+<i>i</i>)) is not equal to 0, (0 ≤ <i>i</i> &lt; 4), <i>i</i>-th 4x4 AC <i>U</i>-block is not zero-filled and exists in <i>ppSrcCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+<i>i</i>+4)) is not equal to 0, (0 ≤ <i>i</i> &lt; 4), <i>i</i>-th 4x4 AC <i>U</i>-block is not zero-filled and exists in <i>ppSrcCoeff</i>.</p>

<i>ChromaQP</i>	Chroma quantizer ( $QP_C$ in [JVTG050]). It must be within the range [0;39].
<i>edgeType</i>	<p>Flag that specifies the availability of the macroblocks used for prediction.</p> <p>If the upper macroblock is not available for 16x16 Intra prediction, <code>edgeType&amp;IPPVC_TOP_EDGE</code> must be non-zero.</p> <p>If the left macroblock is not available for 16x16 Intra prediction, <code>edgeType&amp;IPPVC_LEFT_EDGE</code> must be non-zero.</p> <p>If the upper-left macroblock is not available for 16x16 Intra prediction, <code>edgeType&amp;IPPVC_TOP_LEFT_EDGE</code> must be non-zero.</p>

## Description

The function `ippiReconstructChromaIntraMB_H264_16s8u_P2R` is declared in the `ippvc.h` file. This function reconstructs Intra Chroma macroblock (8x8 *U* macroblock and 8x8 *V* macroblock):

1. Performs integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.7 of [JVTG050] in the same way as [TransformDequantChromaDC\\_H264](#) function does.
2. Performs scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.8 of [JVTG050] in the same way as [DequantTransformResidual\\_H264](#) function does. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7 of [JVTG050].
3. Performs intra prediction for an 8x8 *U*-component and for an 8x8 *V*-component in accordance with 8.3.2 of [JVTG050] in the same way as [PredictIntraChroma8x8\\_H264](#) function does.
4. Performs adding of 8x8 intra prediction blocks and 8x8 residual blocks in accordance with 8-247 of [JVTG050].

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>ChromaQP</i> is less than 0 or greater than 39.

---

## ReconstructChromaInter4x4MB\_H264

*Reconstructs 4X4 Inter Chroma macroblock for high profile.*

---

### Syntax

```
IppStatus ippiReconstructChromaInter4x4MB_H264_16s8u_P2R(Ipp16s  
    **ppSrcDstCoeff, Ipp8u *pSrcDstUPlane, Ipp8u *pSrcDstVPlane, Ipp32u  
    srcDstUVStep, Ipp32u cbp4x4, Ipp32s ChromaQPU, Ipp32u ChromaQPV, Ipp16s  
    *pQuantTableU, Ipp16s *pQuantTableV, Ipp8u bypassFlag);
```

### Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC <i>U</i> -block, 2x2 DC <i>V</i> -block, 4x4 AC <i>U</i> -blocks, 4x4 AC <i>V</i> -blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of <a href="#">[ITUH264]</a> . Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to macroblock that is reconstructed in current <i>U</i> plane. This macroblock must contain inter prediction samples.
<i>pSrcDstVPlane</i>	Pointer to macroblock that is reconstructed in current <i>V</i> plane. This macroblock must contain inter prediction samples.
<i>srcDstUVStep</i>	Plane step.
<i>cbp4x4</i>	<p>Coded block pattern.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)) is not equal to 0, 2x2 DC <i>U</i>-block is not zero-filled and exists in <i>ppSrcDstCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1)) is not equal to 0, 2x2 DC <i>U</i>-block is not zero-filled and exists in <i>ppSrcDstCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+<i>i</i>)) is not equal to 0, (0 ≤ <i>i</i> &lt; 4), <i>i</i>-th 4x4 AC <i>U</i>-block is not zero-filled and exists in <i>ppSrcDstCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+<i>i</i>+4)) is not equal to 0, (0 ≤ <i>i</i> &lt; 4), <i>i</i>-th 4x4 AC <i>U</i>-block is not zero-filled and exists in <i>ppSrcDstCoeff</i>.</p>



<i>ChromaQPU</i>	Chroma quantizer for $U$ plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<i>ChromaQPV</i>	Chroma quantizer for $V$ plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<i>pQuantTableU</i>	Pointer to the quantization table for $U$ plane ( $LevelScale(qP\%6, i, j)$ in [ITUH264]).
<i>pQuantTableV</i>	Pointer to the quantization table for $V$ plane ( $LevelScale(qP\%6, i, j)$ in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding (reserved for future use).

## Description

The function `ippiReconstructChromaInter4x4MB_H264_16s8u_P2R` is declared in the `ippvc.h` file. This function reconstructs 4x4 Inter Chroma macroblock (8x8  $U$  macroblock and 8x8  $V$  macroblock) for high profile:

1. Performs integer inverse transformation and dequantization for 2x2  $U$  DC coefficients and for 2x2  $V$  DC coefficients in accordance with 8.5.9 of [ITUH264] according to 8-323 when `chroma_format_idc` is equal to 1.
2. Performs scaling, integer inverse transformation, and shift for 4x4 AC  $U$ -blocks and shift for 4x4 AC  $V$ -blocks in accordance with 8.5.10 of [ITUH264]. This process is performed on all 4x4 chroma blocks in the same order as is shown in Figure 8-7(a) of [ITUH264] when `chroma_format_idc` is equal to 1.
3. Performs adding of 4x4 inter prediction blocks and 4x4 residual blocks in accordance with 8-306 of [ITUH264].

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>ChromaQPU</i> or <i>ChromaQPV</i> is less than 0 or greater than 39.

---

## ReconstructChromaIntraHalves4x4MB\_H264

*Reconstructs two independent halves of 4X4 Intra Chroma macroblock for high profile.*

---

### Syntax

```
IppStatus ippiReconstructChromaIntraHalves4x4MB_H264_16s8u_P2R(Ipp16s  
    **ppSrcDstCoeff, Ipp8u *pSrcDstUPlane, Ipp8u *pSrcDstVPlane, Ipp32u  
    srcDstUVStep, IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u cbp4x4,  
    Ipp32s ChromaQPU, Ipp32u ChromaQPV, Ipp8u edgeTypeTop, Ipp8u edgeTypeBottom,  
    Ipp16s *pQuantTableU, Ipp16s *pQuantTableV, Ipp8u bypassFlag);
```

### Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC <i>U</i> -block, 2x2 DC <i>V</i> -block, 4x4 AC <i>U</i> -blocks, 4x4 AC <i>V</i> -blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of <a href="#">[ITUH264]</a> . Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to current <i>U</i> plane that is reconstructed.
<i>pSrcDstVPlane</i>	Pointer to current <i>V</i> plane that is reconstructed.
<i>srcDstUVStep</i>	Plane step.
<i>intraChromaMode</i>	Prediction mode of the Intra_chroma prediction process for chroma samples. This mode is defined in the same way as in <a href="#">PredictIntraChroma8x8_H264</a> function.
<i>cbp4x4</i>	<p>Coded block pattern.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)) is not equal to 0, 2x2 DC <i>U</i>-block is not zero-filled and exists in <i>ppSrcDstCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1)) is not equal to 0, 2x2 DC <i>U</i>-block is not zero-filled and exists in <i>ppSrcDstCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+<i>i</i>)) is not equal to 0, (0 ≤ <i>i</i> &lt; 4), <i>i</i>-th 4x4 AC <i>U</i>-block is not zero-filled and exists in <i>ppSrcDstCoeff</i>.</p>

	If <code>cbp4x4</code> & $(1 < (IPPVC\_CBP\_1ST\_CHROMA\_AC\_BITPOS + i + 4))$ is not equal to 0, $(0 \leq i < 4)$ , $i$ -th 4x4 AC $U$ -block is not zero-filled and exists in <code>ppSrcDstCoeff</code> .
<code>ChromaQPU</code>	Chroma quantizer for $U$ plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<code>ChromaQPV</code>	Chroma quantizer for $V$ plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<code>edgeTop,</code> <code>edgeTypeBottom</code>	Flags that specify the availability of the macroblocks used for prediction. Upper and lower halves of the macroblock may have different prediction vectors, so use two flags: one for the upper half of the macroblock, the other - for the lower half.
<code>pQuantTableU</code>	Pointer to the quantization table for $U$ plane ( $LevelScale(qP\%6, i, j)$ in [ITUH264]).
<code>pQuantTableV</code>	Pointer to the quantization table for $V$ plane ( $LevelScale(qP\%6, i, j)$ in [ITUH264]).
<code>bypassFlag</code>	Flag enabling lossless coding (reserved for future use).

## Description

The function `ippiReconstructChromaIntraHalves4x4MB_H264_16s8u_P2R` is declared in the `ippvc.h` file. This function reconstructs 4x4 Intra Chroma macroblock (8x8  $U$  macroblock and 8x8  $V$  macroblock) for high profile. The macroblock is divided into upper and lower halves, 16x8 each. The halves differ only in prediction, that is, have independent prediction vectors. Otherwise their performance is the same:

1. Performs integer inverse transformation and dequantization for 2x2  $U$  DC coefficients and for 2x2  $V$  DC coefficients in accordance with 8.5.9 of [ITUH264] according to 8-323 when `chroma_format_idc` is equal to 1.
2. Performs scaling, integer inverse transformation, and shift for 4x4 AC  $U$ -blocks and shift for 4x4 AC  $V$ -blocks in accordance with 8.5.10 of [ITUH264]. This process is performed on all 4x4 chroma blocks in the same order as shown in Figure 8-7(a) of [ITUH264] when `chroma_format_idc` is equal to 1.
3. Performs adding of 4x4 inter prediction blocks and 4x4 residual blocks in accordance with 8-306 of [ITUH264].

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>ChromaQPU</i> or <i>ChromaQPV</i> is less than 0 or greater than 39.

---

## ReconstructChromaIntra4x4MB\_H264

*Reconstructs 4X4 Intra Chroma macroblock for high profile.*

---

### Syntax

```
IppStatus ippReconstructChromaIntra4x4MB_H264_16s8u_P2R(Ipp16s
**ppSrcDstCoeff, Ipp8u *pSrcDstUPlane, Ipp8u *pSrcDstVPlane, Ipp32u
srcDstUVStep, IppIntraChromaPredMode_H264 intraChromaMode, Ipp32u cbp4x4,
Ipp32s ChromaQPU, Ipp32u ChromaQPV, Ipp8u edgeType, Ipp16s *pQuantTableU,
Ipp16s *pQuantTableV, Ipp8u bypassFlag);
```

### Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (2x2 DC <i>U</i> -block, 2x2 DC <i>V</i> -block, 4x4 AC <i>U</i> -blocks, 4x4 AC <i>V</i> -blocks if the block is not zero-filled) in the same order as is shown in Figure 8-7 of <a href="#">[ITUH264]</a> . Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstUPlane</i>	Pointer to macroblock that is reconstructed in current <i>U</i> plane. This macroblock must contain inter prediction samples.
<i>pSrcDstVPlane</i>	Pointer to macroblock that is reconstructed in current <i>V</i> plane. This macroblock must contain inter prediction samples.
<i>srcDstUVStep</i>	Plane step.
<i>intraChromaMode</i>	Prediction mode of the Intra_chroma prediction process for chroma samples. This mode is defined in the same way as in <a href="#">PredictIntraChroma8x8_H264</a> function.

<i>cbp4x4</i>	<p>Coded block pattern.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_DC_BITPOS)) is not equal to 0, 2x2 DC <i>U</i>-block is not zero-filled and exists in <i>ppSrcDstCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_DC_BITPOS+1)) is not equal to 0, 2x2 DC <i>U</i>-block is not zero-filled and exists in <i>ppSrcDstCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i)) is not equal to 0, (0 ≤ <i>i</i> &lt; 4), <i>i</i>-th 4x4 AC <i>U</i>-block is not zero-filled and exists in <i>ppSrcDstCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1&lt;&lt;(IPPVC_CBP_1ST_CHROMA_AC_BITPOS+i+4)) is not equal to 0, (0 ≤ <i>i</i> &lt; 4), <i>i</i>-th 4x4 AC <i>U</i>-block is not zero-filled and exists in <i>ppSrcDstCoeff</i>.</p>
<i>ChromaQPU</i>	Chroma quantizer for <i>U</i> plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<i>ChromaQPV</i>	Chroma quantizer for <i>V</i> plane ( $QP_C$ in [ITUH264]). It must be within the range [0;39].
<i>edgeType</i>	<p>Flag that specifies the availability of the macroblocks used for prediction.</p> <p>If the upper macroblock is not available for 16x16 Intra prediction, <i>edgeType</i>&amp;IPPVC_TOP_EDGE must be non-zero.</p> <p>If the left macroblock is not available for 16x16 Intra prediction, <i>edgeType</i>&amp;IPPVC_LEFT_EDGE must be non-zero.</p> <p>If the upper-left macroblock is not available for 16x16 Intra prediction, <i>edgeType</i>&amp;IPPVC_TOP_LEFT_EDGE must be non-zero.</p>
<i>pQuantTableU</i>	Pointer to the quantization table for <i>U</i> plane ( <i>LevelScale</i> ( <i>qP</i> %6, <i>i</i> , <i>j</i> ) in [ITUH264]).
<i>pQuantTableV</i>	Pointer to the quantization table for <i>V</i> plane ( <i>LevelScale</i> ( <i>qP</i> %6, <i>i</i> , <i>j</i> ) in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding (reserved for future use).

## Description

The function `ippiReconstructChromaIntra4x4MB_H264_16s8u_P2R` is declared in the `ippvc.h` file. This function reconstructs 4x4 Intra Chroma macroblock (8x8 *U* macroblock and 8x8 *V* macroblock) for high profile:

1. Performs integer inverse transformation and dequantization for 2x2 *U* DC coefficients and for 2x2 *V* DC coefficients in accordance with 8.5.9 of [ITUH264] according to 8-323 when `chroma_format_idc` is equal to 1.
2. Performs scaling, integer inverse transformation, and shift for 4x4 AC *U*-blocks and shift for 4x4 AC *V*-blocks in accordance with 8.5.10 of [ITUH264]. This process is performed on all 4x4 chroma blocks in the same order as shown in Figure 8-7(a) of [ITUH264] when `chroma_format_idc` is equal to 1.
3. Performs adding of 4x4 inter prediction blocks and 4x4 residual blocks in accordance with 8-306 of [ITUH264].

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<code>ChromaQPU</code> or <code>ChromaQPV</code> is less than 0 or greater than 39.

---

## ReconstructLumaInterMB\_H264

*Reconstructs Inter Luma macroblock.*

---

### Syntax

```
IppStatus ippReconstructLumaInterMB_H264_16s8u_C1R(Ipp16s **ppSrcCoeff, Ipp8u
    *pSrcDstYPlane, Ipp32u srcDstYStep, Ipp32u cbp4x4, Ipp32s QP);
```

### Parameters

<code>ppSrcCoeff</code>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 Luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-6 of [JVTG050]. Pointer is updated by the function and points to the blocks for the next macroblock.
<code>pSrcDstYPlane</code>	Pointer to the current macroblock that is reconstructed in current <i>Y</i> -plane. This macroblock must contain inter prediction samples.

<i>srcDstStep</i>	Plane step.
<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & (1 << (1+i)) is not equal to 0 ( $0 \leq i < 16$ ), <i>i</i> -th 4x4 AC Luma block is not zero-filled and it exists in <i>ppSrcCoeff</i> .
<i>QP</i>	Quantization parameter ( $QP_Y$ in [JVTG050]). It must be within the range [0;51].

## Description

The function `ippiReconstructLumaInterMB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs Inter Luma macroblock:

1. Performs scaling, integer inverse transformation, and shift for each 4x4 block in the same order as is shown in Figure 6-6 of [JVTG050] in accordance with 8.5.8 of [JVTG050] in the same way as `DequantTransformResidual_H264` function does.
2. Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 8-247 of [JVTG050].

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 51.

---

## ReconstructLumaIntraHalfMB\_H264

*Reconstructs half of Intra Luma macroblock.*

---

## Syntax

```
IppStatus ippiReconstructLumaIntraHalfMB_H264_16s8u_C1R(Ipp16s **ppSrcCoeff,
    Ipp8u *pSrcDstYPlane, Ipp32s srcDstYStep, IppIntra4x4PredMode_H264
    *pMBIntraTypes, Ipp32u cbp4x2, Ipp32u QP, Ipp8u edgeType);
```

## Parameters

<i>ppSrcCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 Luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-6 of <a href="#">[JVTG050]</a> . Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in Y-plane.
<i>srcDstYStep</i>	Y-Plane step.
<i>pMBIntraTypes</i>	Array of Intra_4x4 luma prediction modes for 8 subblocks. <i>pMBIntraTypes[i]</i> is defined in the same way as in <a href="#">PredictIntra 4x4 H264</a> function ( $0 \leq i < 8$ ).
<i>cbp4x2</i>	Coded block pattern. If <i>cbp4x2</i> & (1<<(1+i)) is not equal to 0 ( $0 \leq i < 8$ ), <i>i</i> -th 4x4 AC Luma block is not zero-filled and it exists in <i>ppSrcCoeff</i> .
<i>QP</i>	Quantization parameter ( $QP_Y$ in <a href="#">[JVTG050]</a> ). It must be within the range [0;51].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 4x4 Intra prediction, <i>edgeType&amp;IPPVC_TOP_EDGE</i> must be non-zero. If the left macroblock is not available for 4x4 Intra prediction, <i>edgeType&amp;IPPVC_LEFT_EDGE</i> must be non-zero. If the upper-left macroblock is not available for 4x4 Intra prediction, <i>edgeType&amp;IPPVC_TOP_LEFT_EDGE</i> must be non-zero. If the upper-right macroblock is not available for 4x4 Intra prediction, <i>edgeType&amp;IPPVC_TOP_RIGHT_EDGE</i> must be non-zero.

## Description

The function `ippiReconstructLumaIntraMB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs a half of Intra Luma macroblock, that is, eight 4x4 subblocks. The process for the eight 4x4 blocks (from 0 to 7 or from 8 to 15) in the same order as is shown in Figure 6-6 of [\[JVTG050\]](#) is described below:



1. Performs scaling, integer inverse transformation, and shift for a 4x4 block in accordance with 8.5.8 of [\[JVTG050\]](#) in the same way as [DequantTransformResidual\\_H264](#) function does.
2. Performs intra prediction for a 4x4 luma component in accordance with 8.3.1.2 of [\[JVTG050\]](#) in the same way as [PredictIntra\\_4x4\\_H264](#) function does.
3. Performs adding of 16x8 prediction block and 16x8 residual block in accordance with 8-247 of [\[JVTG050\]](#).

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 51.

---

## ReconstructLumaIntraMB\_H264

*Reconstructs Intra Luma macroblock.*

---

### Syntax

```
IppStatus ippReconstructLumaIntraMB_H264_16s8u_C1R(Ipp16s **ppSrcCoeff, Ipp8u
*pSrcDstYPlane, Ipp32s srcDstYStep, const IppIntra4x4PredMode_H264
*pMBIntraTypes, const Ipp32u cbp4x4, const Ipp32u QP, const Ipp8u edgeType);
```

### Parameters

<i>ppSrcCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 Luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-6 of <a href="#">[JVTG050]</a> . Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in Y-plane.
<i>srcDstYStep</i>	Y-Plane step.
<i>pMBIntraTypes</i>	Array of <code>Intra_4x4</code> luma prediction modes for each subblock. <code>pMBIntraTypes[i]</code> is defined in the same way as in <a href="#">PredictIntra_4x4_H264</a> function ( $0 \leq i < 16$ ).

<i>cbp4x4</i>	<p>Coded block pattern.</p> <p>If <i>cbp4x4</i> &amp; (1 &lt;&lt; (1 + <i>i</i>)) is not equal to 0 (<math>0 \leq i &lt; 16</math>), <i>i</i>-th 4x4 AC Luma block is not zero-filled and it exists in <i>ppSrcCoeff</i>.</p>
<i>QP</i>	Quantization parameter ( <i>QP<sub>Y</sub></i> in [JVTG050]). It must be within the range [0;51].
<i>edgeType</i>	<p>Flag that specifies the availability of the macroblocks used for prediction.</p> <p>If the upper macroblock is not available for 4x4 Intra prediction, <i>edgeType</i>&amp;IPPVC_TOP_EDGE must be non-zero.</p> <p>If the left macroblock is not available for 4x4 Intra prediction, <i>edgeType</i>&amp;IPPVC_LEFT_EDGE must be non-zero.</p> <p>If the upper-left macroblock is not available for 4x4 Intra prediction, <i>edgeType</i>&amp;IPPVC_TOP_LEFT_EDGE must be non-zero.</p> <p>If the upper-right macroblock is not available for 4x4 Intra prediction, <i>edgeType</i>&amp;IPPVC_TOP_RIGHT_EDGE must be non-zero.</p>

## Description

The function `ippiReconstructLumaIntraMB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs Intra Luma macroblock. The process for each 4x4 block in the same order as is shown in Figure 6-6 of [JVTG050] is described below:

1. Performs scaling, integer inverse transformation, and shift for a 4x4 block in accordance with 8.5.8 of [JVTG050] in the same way as [DequantTransformResidual\\_H264](#) function does.
2. Performs intra prediction for a 4x4 luma component in accordance with 8.3.1.2 of [JVTG050] in the same way as [PredictIntra\\_4x4\\_H264](#) function does.
3. Performs adding of 4x4 prediction block and 4x4 residual block in accordance with 8-247 of [JVTG050].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 51.

---

## ReconstructLumaInter4x4MB\_H264

*Reconstructs 4X4 Inter Luma macroblock for high profile.*

---

### Syntax

```
IppStatus ippiReconstructLumaInter4x4MB_H264_16s8u_C1R(Ipp16s **ppSrcDstCoeff,
    Ipp8u *pSrcDstYPlane, Ipp32u srcDstYStep, Ipp32u cbp4x4, Ipp32s QP, Ipp16s
    *pQuantTable, Ipp8u bypassFlag);
```

### Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 Luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-10 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in current Y-plane. This macroblock must contain inter prediction samples.
<i>srcDstStep</i>	Plane step.
<i>cbp4x4</i>	Coded block pattern. If $cbp4x4 \ \& \ (1 < (1+i))$ is not equal to 0 ( $0 \leq i < 16$ ), $i$ -th 4x4 AC Luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> .
<i>QP</i>	Quantization parameter ( $QP_Y$ in [ITUH264]). It must be within the range [0;51].
<i>pQuantTable</i>	Pointer to the quantization table ( $LevelScale(qP\%6, i, j)$ in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding (reserved for future use).

### Description

The function `ippiReconstructLumaInter4x4MB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs a 4x4 Inter Luma macroblock for high profile:

1. Performs scaling, integer inverse transformation, and shift for each 4x4 block in the same order as is shown in Figure 6-10 of [ITUH264] in accordance with 8.5.10 of [ITUH264].
2. Performs adding of sixteen 4x4 inter prediction blocks and 4x4 residual blocks in accordance with 8-295 of [ITUH264].

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 51.

---

## ReconstructLumaIntraHalf4x4MB\_H264

*Reconstructs half of 4X4 Intra Luma macroblock for high profile.*

---

### Syntax

```
IppStatus ippiReconstructLumaIntraHalf4x4MB_H264_16s8u_C1R(Ipp16s
**ppSrcDstCoeff, Ipp8u *pSrcDstYPlane, Ipp32s srcDstYStep,,
IppIntra4x4PredMode_H264 *pMBIntraTypes, Ipp32u cbp4x2, Ipp32s QP, Ipp8u
edgeType, Ipp16s *pQuantTable, Ipp8u bypassFlag);
```

### Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 Luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-10 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in Y-plane.
<i>srcDstYStep</i>	Y-plane step.
<i>pMBIntraTypes</i>	Array of Intra_4x4 luma prediction modes for eight subblocks. <i>pMBIntraTypes[i]</i> is defined in the same way as in <a href="#">PredictIntra_4x4_H264</a> function ( $0 \leq i < 8$ ).
<i>cbp4x2</i>	Coded block pattern. If <i>cbp4x2</i> & $(1 < (1+i))$ is not equal to 0 ( $0 \leq i < 8$ ), <i>i</i> -th 4x4 AC Luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> .

<i>QP</i>	Quantization parameter ( $QP_Y$ in [ITUH264]). It must be within the range [0;51].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 4x4 Intra prediction, <code>edgeType&amp;IPPVC_TOP_EDGE</code> must be non-zero. If the left macroblock is not available for 4x4 Intra prediction, <code>edgeType&amp;IPPVC_LEFT_EDGE</code> must be non-zero. If the upper-left macroblock is not available for 4x4 Intra prediction, <code>edgeType&amp;IPPVC_TOP_LEFT_EDGE</code> must be non-zero. If the upper-right macroblock is not available for 4x4 Intra prediction, <code>edgeType&amp;IPPVC_TOP_RIGHT_EDGE</code> must be non-zero.
<i>pQuantTable</i>	Pointer to the quantization table ( $LevelScale(qP\%6, i, j)$ in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding (reserved for future use).

## Description

The function `ippiReconstructLumaIntraHalf4x4MB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs half of an Intra Luma macroblock for high profile. The process for eight 4x4 blocks (from 0 to 7 or from 8 to 15) in the same order as is shown in Figure 6-10 of [ITUH264] is described below:

1. Performs scaling, integer inverse transformation, and shift for 4x4 blocks in accordance with 8.5.10 of [ITUH264].
2. Performs intra prediction for 4x4 luma components in accordance with 8.3.1.2 of [ITUH264].
3. Performs adding of a 16x8 prediction block (eight 4x4 blocks) and a 16x8 residual block in accordance with 8-295 of [ITUH264].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	$QP$ is less than 0 or greater than 51.

## ReconstructLumaIntra4x4MB\_H264

*Reconstructs 4X4 Intra Luma macroblock for high profile.*

---

### Syntax

```
IppStatus ippiReconstructLumaIntra4x4MB_H264_16s8u_C1R(Ipp16s **ppSrcDstCoeff,
Ipp8u *pSrcDstYPlane, Ipp32s srcDstYStep,, IppIntra4x4PredMode_H264
*pMBIntraTypes, Ipp32u cbp4x4, Ipp32s QP, Ipp8u edgeType, Ipp16s
*pQuantTable, Ipp8u bypassFlag);
```

### Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 Luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-10 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in current Y-plane.
<i>srcDstYStep</i>	Y-plane step.
<i>pMBIntraTypes</i>	Array of Intra_4x4 luma prediction modes for each subblock. <i>pMBIntraTypes[i]</i> is defined in the same way as in <a href="#">PredictIntra_4x4_H264</a> function ( $0 \leq i < 16$ ).
<i>cbp4x4</i>	Coded block pattern. If <i>cbp4x4</i> & (1<<(1+i)) is not equal to 0 ( $0 \leq i < 16$ ), <i>i</i> -th 4x4 AC Luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> .
<i>QP</i>	Quantization parameter ( $QP_Y$ in [ITUH264]). It must be within the range [0;51].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 4x4 Intra prediction, <i>edgeType</i> &IPPVC_TOP_EDGE must be non-zero. If the left macroblock is not available for 4x4 Intra prediction, <i>edgeType</i> &IPPVC_LEFT_EDGE must be non-zero. If the upper-left macroblock is not available for 4x4 Intra prediction,

	edgeType&IPPVC_TOP_LEFT_EDGE must be non-zero. If the upper-right macroblock is not available for 4x4 Intra prediction, edgeType&IPPVC_TOP_RIGHT_EDGE must be non-zero.
<i>pQuantTable</i>	Pointer to the quantization table ( <i>LevelScale(qP%6, i, j)</i> in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding (reserved for future use).

## Description

The function `ippiReconstructLumaIntra4x4MB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs Inter Luma macroblock for high profile. The process for each 4x4 block in the same order as is shown in Figure 6-10 of [ITUH264] is described below:

1. Performs scaling, integer inverse transformation, and shift for each 4x4 block in accordance with 8.5.10 of [ITUH264].
2. Performs intra prediction for a 4x4 luma component in accordance with 8.3.1.2 of [ITUH264].
3. Performs adding of sixteen 4x4 prediction blocks and sixteen 4x4 residual blocks in accordance with 8-295 of [ITUH264].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 51.

---

## ReconstructLumaInter8x8MB\_H264

*Reconstructs 8X8 Inter Luma macroblock for high profile.*

---

## Syntax

```
IppStatus ippiReconstructLumaInter8x8MB_H264_16s8u_C1R(Ipp16s **ppSrcDstCoeff,
    Ipp8u *pSrcDstYPlane, Ipp32u srcDstYStep, Ipp32u cbp8x8, Ipp32s QP, Ipp16s
    *pQuantTable, Ipp8u bypassFlag);
```

## Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (8x8 Luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-11 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in current Y-plane. This macroblock must contain inter prediction samples.
<i>srcDstStep</i>	Plane step.
<i>cbp8x8</i>	Coded block pattern. If $cbp8x8 \ \& \ (1 < (1+i))$ is not equal to 0 ( $0 \leq i < 4$ ), $i$ -th 8x8 AC Luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> .
<i>QP</i>	Quantization parameter ( $QP_Y$ in [ITUH264]). It must be within the range [0;51].
<i>pQuantTable</i>	Pointer to the quantization table ( $LevelScale(qP\%6, i, j)$ in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding (reserved for future use).

## Description

The function `ippiReconstructLumaInter8x8MB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs 8x8 Inter Luma macroblock for high profile:

1. Performs scaling, integer inverse transformation, and shift for each 8x8 block in the same order as is shown in Figure 6-11 of [ITUH264] in accordance with 8.5.11 of [ITUH264].
2. Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 8-298 of [ITUH264].

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<i>ippStsNoErr</i>	Indicates no error.
<i>ippStsNullPtrErr</i>	Indicates an error condition if at least one of the specified pointers is NULL.
<i>ippStsOutOfRangeErr</i>	<i>QP</i> is less than 0 or greater than 51.



## ReconstructLumaIntraHalf8x8MB\_H264

*Reconstructs half of 8X8 Intra Luma macroblock for high profile.*

### Syntax

```
IppStatus ippiReconstructLumaIntraHalf8x8MB_H264_16s8u_C1R(Ipp16s
    **ppSrcDstCoeff, Ipp8u *pSrcDstYPlane, Ipp32s srcDstYStep,
    IppIntra8x8PredMode_H264 *pMBOIntraTypes, Ipp32u cbp8x2, Ipp32s QP, Ipp8u
    edgeType, Ipp16s *pQuantTable, Ipp8u bypassFlag);
```

### Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (8x8 Luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-11 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in Y-plane.
<i>srcDstStep</i>	Y-plane step.
<i>pMBOIntraTypes</i>	Array of Intra_8x8 luma prediction modes for each subblock. <i>pMBOIntraTypes[i]</i> is defined for subblocks ( $0 \leq i < 2$ ) as described in 8.3.2.1 of [ITUH264].
<i>cbp8x2</i>	Coded block pattern. If <i>cbp8x2</i> & (1<<(1+i)) is not equal to 0 ( $0 \leq i < 2$ ), <i>i</i> -th 8x8 AC Luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> .
<i>QP</i>	Quantization parameter ( $QP_Y$ in [ITUH264]). It must be within the range [0;51].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 Intra prediction, <i>edgeType</i> &IPPVC_TOP_EDGE must be non-zero. If the left macroblock is not available for 8x8 Intra prediction, <i>edgeType</i> &IPPVC_LEFT_EDGE must be non-zero. If the upper-left macroblock is not available for 8x8 Intra prediction,

edgeType&IPPVC\_TOP\_LEFT\_EDGE must be non-zero.

If the upper-right macroblock is not available for 8x8 Intra prediction, edgeType&IPPVC\_TOP\_RIGHT\_EDGE must be non-zero.

<i>pQuantTable</i>	Pointer to the quantization table ( <i>LevelScale(qP%6, i, j)</i> in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding (reserved for future use).

## Description

The function `ippiReconstructLumaIntraHalf8x8MB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs 8x8 Intra Luma macroblock for high profile. The process for each 8x8 block in the same order as shown in Figure 6-11 of [ITUH264] is described below:

1. Performs scaling, integer inverse transformation, and shift for a 8x8 block in accordance with 8.5.11 of [ITUH264].
2. Performs intra prediction for a 8x8 luma component in accordance with 8.3.2 of [ITUH264].
3. Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 8-298 of [ITUH264].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 51.

---

## ReconstructLumaIntra8x8MB\_H264

*Reconstructs 8X8 Intra Luma macroblock for high profile.*

---

## Syntax

```
IppStatus ippiReconstructLumaIntra8x8MB_H264_16s8u_C1R(Ipp16s **ppSrcDstCoeff,
    Ipp8u *pSrcDstYPlane, Ipp32s srcDstYStep, IppIntra8x8PredMode_H264
    *pMBOIntraTypes, Ipp32u cbp8x8, Ipp32s QP, Ipp8u edgeType, Ipp16s
    *pQuantTable, Ipp8u bypassFlag);
```

## Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 8x8 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (8x8 Luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-11 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in current Y-plane.
<i>srcDstStep</i>	Y-plane step.
<i>pMBIntraTypes</i>	Array of Intra_8x8 luma prediction modes for each subblock. <i>pMBIntraTypes[i]</i> is defined for subblocks ( $0 \leq i < 4$ ) as described in 8.3.2.1 of [ITUH264].
<i>cbp8x8</i>	Coded block pattern. If <i>cbp8x8</i> & (1<<(1+i)) is not equal to 0 ( $0 \leq i < 4$ ), <i>i</i> -th 8x8 AC Luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> .
<i>QP</i>	Quantization parameter ( $QP_Y$ in [ITUH264]). It must be within the range [0;51].
<i>edgeType</i>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 8x8 Intra prediction, <i>edgeType&amp;IPPVC_TOP_EDGE</i> must be non-zero. If the left macroblock is not available for 8x8 Intra prediction, <i>edgeType&amp;IPPVC_LEFT_EDGE</i> must be non-zero. If the upper-left macroblock is not available for 8x8 Intra prediction, <i>edgeType&amp;IPPVC_TOP_LEFT_EDGE</i> must be non-zero. If the upper-right macroblock is not available for 8x8 Intra prediction, <i>edgeType&amp;IPPVC_TOP_RIGHT_EDGE</i> must be non-zero.
<i>pQuantTable</i>	Pointer to the quantization table ( <i>LevelScale</i> ( $qP\%6, i, j$ ) in [ITUH264]).
<i>bypassFlag</i>	Flag enabling lossless coding (reserved for future use).

## Description

The function `ippiReconstructLumaIntra8x8MB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs 8x8 Intra Luma macroblock for high profile. The process for each 8x8 block in the same order as shown in Figure 6-11 of [ITUH264] is described below:

1. Performs scaling, integer inverse transformation, and shift for a 8x8 block in accordance with 8.5.11 of [\[ITUH264\]](#).
2. Performs intra prediction for a 8x8 luma component in accordance with 8.3.2 of [\[ITUH264\]](#).
3. Performs adding of a 16x16 inter prediction block and a 16x16 residual block in accordance with 8-298 of [\[ITUH264\]](#).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	<i>QP</i> is less than 0 or greater than 51.

---

## ReconstructLumaIntra16x16MB\_H264

*Reconstructs 16X16 Intra Luma macroblock.*

---

## Syntax

```
IppStatus ippReconstructLumaIntra16x16MB_H264_16s8u_C1R(Ipp16s **ppSrcCoeff,
Ipp8u *pSrcDstYPlane, Ipp32u srcDstYStep, const IppIntra16x16PredMode_H264
intraLumaMode, const Ipp32u cbp4x4, const Ipp32u QP, const Ipp8u edgeType);
```

## Parameters

<i>ppSrcCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 DC Luma blocks, 4x4 AC Luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-6 of <a href="#">[JVTG050]</a> . Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in Y-plane.
<i>srcDstYStep</i>	Y-Plane step.
<i>intraLumaMode</i>	Prediction mode of the Intra_16x16 prediction process for luma samples. It is defined in the same way as in <a href="#">PredictIntra 16x16 H264</a> function.

<i>cbp4x4</i>	<p>Coded block pattern.</p> <p>If <i>cbp4x4</i> &amp; IPPVC_CBP_LUMA_DC is not equal to 0, 4x4 DC Luma block is not zero-filled and it exists in <i>ppSrcCoeff</i>.</p> <p>If <i>cbp4x4</i> &amp; (1 &lt; (IPPVC_CBP_1ST_LUMA_AC_BITPOS + <i>i</i>)) is not equal to 0 (<math>0 \leq i &lt; 16</math>), <i>i</i>-th 4x4 AC Luma block is not zero-filled and it exists in <i>ppSrcCoeff</i>.</p>
<i>QP</i>	Quantization parameter ( $QP_Y$ in [JVTG050]). It must be within the range [0;51].
<i>edgeType</i>	<p>Flag that specifies the availability of the macroblocks used for prediction.</p> <p>If the upper macroblock is not available for 16x16 Intra prediction, <i>edgeType</i>&amp;IPPVC_TOP_EDGE must be non-zero.</p> <p>If the left macroblock is not available for 16x16 Intra prediction, <i>edgeType</i>&amp;IPPVC_LEFT_EDGE must be non-zero.</p> <p>If the upper-left macroblock is not available for 16x16 Intra prediction, <i>edgeType</i>&amp;IPPVC_TOP_LEFT_EDGE must be non-zero.</p>

## Description

The function `ippiReconstructLumaIntra16x16MB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs 16x16 Intra Luma macroblock:

1. Performs integer inverse transformation and dequantization for 4x4 Luma DC coefficients in accordance with 8.5.6 of [JVTG050] in the same way as [TransformDequantLumaDC\\_H264](#) function does.
2. Performs scaling, integer inverse transformation, and shift for 4x4 AC blocks in accordance with 8.5.8 of [JVTG050] in the same way as [DequantTransformResidual\\_H264](#) function does. This process is performed on all 4x4 AC blocks in the same order as Figure 6-6 of [JVTG050] shows.
3. Performs intra prediction for a 16x16 luma component in accordance with 8.3.2 of [JVTG050] in the same way as [PredictIntra\\_16x16\\_H264](#) function does.
4. Performs adding of 16x16 prediction blocks and 16x16 residual blocks in accordance with 8-247 of [JVTG050].

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

`ippStsNoErr` Indicates no error.

`ippStsNullPtrErr` Indicates an error condition if at least one of the specified pointers is NULL.

`ippStsOutOfRangeErr`  $QP$  is less than 0 or greater than 51.

---

## ReconstructLumaIntra\_16x16MB\_H264

*Reconstructs 16X16 Intra Luma macroblock for high profile.*

---

### Syntax

```
IppStatus ippIReconstructLumaIntra_16x16MB_H264_16s8u_C1R(Ipp16s
**ppSrcDstCoeff, Ipp8u *pSrcDstYPlane, Ipp32u srcDstYStep,
IppIntra16x16PredMode_H264 intraLumaMode, Ipp32u cbp4x4, Ipp32u QP, Ipp8u
edgeType, Ipp16s *pQuantTable, Ipp8u bypassFlag);
```

### Parameters

<i>ppSrcDstCoeff</i>	Pointer to the order of 4x4 blocks of residual coefficients for this macroblock, which are taken as a result of Huffman decoding (4x4 Luma blocks, if the block is not zero-filled) in the same order as is shown in Figure 6-10 of [ITUH264]. Pointer is updated by the function and points to the blocks for the next macroblock.
<i>pSrcDstYPlane</i>	Pointer to the current macroblock that is reconstructed in Y-plane.
<i>srcDstYStep</i>	Y-Plane step.
<i>intraLumaMode</i>	Prediction mode of the Intra_16x16 prediction process for luma samples. It is defined in the same way as in <a href="#">PredictIntra 16x16 H264</a> function.
<i>cbp4x4</i>	Coded block pattern. If $cbp4x4 \ \& \ IPPVC\_CBP\_LUMA\_DC$ is not equal to 0, 4x4 DC Luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> . If $cbp4x4 \ \& \ (1 < (IPPVC\_CBP\_1ST\_LUMA\_AC\_BITPOS + i))$ is not equal to 0 ( $0 \leq i < 16$ ), $i$ -th 4x4 AC Luma block is not zero-filled and it exists in <i>ppSrcDstCoeff</i> .
<i>QP</i>	Quantization parameter ( $QP_Y$ in [ITUH264]). It must be within the range [0;51].

<code>edgeType</code>	Flag that specifies the availability of the macroblocks used for prediction. If the upper macroblock is not available for 16x16 Intra prediction, <code>edgeType&amp;IPPVC_TOP_EDGE</code> must be non-zero. If the left macroblock is not available for 16x16 Intra prediction, <code>edgeType&amp;IPPVC_LEFT_EDGE</code> must be non-zero. If the upper-left macroblock is not available for 16x16 Intra prediction, <code>edgeType&amp;IPPVC_TOP_LEFT_EDGE</code> must be non-zero.
<code>pQuantTable</code>	Pointer to the quantization table ( <i>LevelScale</i> ( $qP\%6$ , $i$ , $j$ ) in [ITUH264]).
<code>bypassFlag</code>	Flag enabling lossless coding (reserved for future use).

## Description

The function `ippiReconstructLumaIntra_16x16MB_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function reconstructs 16x16 Intra Luma macroblock for high profile:

1. Performs integer inverse transformation and dequantization for 4x4 Luma DC coefficients in accordance with 8.5.8 of [ITUH264].
2. Performs scaling, integer inverse transformation, and shift for 4x4 AC blocks in accordance with 8.5.10 of [ITUH264]. This process is performed on all 4x4 AC blocks in the same order as Figure 6-10 of [ITUH264] shows.
3. Performs intra prediction for a 16x16 luma component in accordance with 8.3.3 of [ITUH264] in the same way as [PredictIntra\\_16x16\\_H264](#) function does.
4. Performs adding of 16x16 prediction blocks and 16x16 residual blocks in accordance with 8-297 of [ITUH264].

This function is used in the H.264 decoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	$QP$ is less than 0 or greater than 51.

## Deblocking Filtering

---

### FilterDeblockingLuma\_VerEdge\_H264

*Performs deblocking filtering on the vertical edges of the 16x16 luma macroblock.*

---

#### Syntax

```
IppStatus ippiFilterDeblockingLuma_VerEdge_H264_8u_C1IR(Ipp8u* pSrcDst,  
    Ipp32s srcdstStep, Ipp8u* pAlpha, Ipp8u* pBeta, Ipp8u* pThresholds, Ipp8u*  
    pBs);
```

#### Parameters

<i>pSrcDst</i>	Pointer to the initial and resultant coefficients.
<i>srcdstStep</i>	Step of the array.
<i>pAlpha</i>	Array of size 2 of Alpha Thresholds (values for external and internal vertical edge).
<i>pBeta</i>	Array of size 2 of Beta Thresholds (values for external and internal vertical edge).
<i>pThresholds</i>	Array of size 16 of Thresholds ( $T_{c0}$ ) (values for the left edge of each 4x4 block).
<i>pBs</i>	Array of size 16 of BS parameters (values for the left edge of each 4x4 block).

#### Description

The function `ippiFilterDeblockingLuma_VerEdge_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs Deblocking Filtering on the vertical edges of the 16x16 luma macroblock in accordance with 8.7.2 of [\[JVTG050\]](#).

The function uses arrays *pAlpha*, *pBeta*, *pBS*, *pThresholds* as input arguments, where *pAlpha*[0], *pBeta*[0] are values for the external vertical edge, and *pAlpha*[1], *pBeta*[1] are values for the internal vertical edge. See [Figure 16-65](#) for the arrangement of *pThresholds* and *pBS* array elements.

Values of the arrays are calculated as follows:



- *pBS* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.
- *pAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].
- *pBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].
- *pThresholds[i]* values are calculated from the formulas 8-326, 8-327, values of *pBS* array and Table 8-15 of [JVTG050].

**Figure 16-65    Arrangement of *pThresholds* Array Elements into a macroblock**

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

This function is used in the H.264 decoder and encoder included into IPP Samples. See [introduction](#) to this section.

**Return Values**

- ippStsNoErr

Indicates no error.
- ippStsNullPtrErr

Indicates an error condition if at least one of the specified pointers is NULL.

## FilterDeblockingLuma\_VerEdge\_MBAFF\_H264

*Performs deblocking filtering on the external vertical edges of half of 16X16 luma macroblock.*

---

### Syntax

```
IppStatus ippiFilterDeblockingLuma_VerEdge_MBAFF_H264_8u_C1IR( Ipp8u*
    pSrcDst, Ipp32s srcdstStep, Ipp32u nAlpha, Ipp32u nBeta, Ipp8u* pThresholds,
    Ipp8u* pBS);
```

### Parameters

<i>pSrcDst</i>	Pointer to the initial and resultant coefficients.
<i>srcdstStep</i>	Step of the array.
<i>pAlpha</i>	Alpha Threshold.
<i>pBeta</i>	Beta Threshold.
<i>pThresholds</i>	Array of size 8 of Thresholds ( $T_{c0}$ ) (two values for the left edge of each 4x4 block).
<i>pBS</i>	Array of size 8 of BS parameters (two values for the left edge of each 4x4 block).

### Description

The function `ippiFilterDeblockingLuma_VerEdge_MBAFF_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs Deblocking Filtering on the vertical edges of the 16x16 luma macroblock in accordance with 8.7.2 of [\[JVTG050\]](#).

In process of filling *pThresholds* and *pBS* parameters for deblocking of each 4x4 block, a number of parameters of the neighboring blocks are taken into account. In MBAFF mode, the edge blocks may border on two others when the left and the current macroblocks are coded differently: one as a field, the other as a frame, and vice versa. As a result, two values of *pThresholds* and *pBS* parameters are taken for each block. For higher flexibility, the function processes only half of the macroblock to fit both types of the macroblocks, that is, coded as a field and coded as a frame.

The function uses arrays *nAlpha*, *nBeta*, *pBS*, *pThresholds* as input arguments. See [Figure 16-65](#) for the arrangement of *pThresholds* and *pBS* array elements.

Values of the arrays are calculated as follows:

- *pBS* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.
- *nAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].
- *nBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].
- *pThresholds[i]* values are calculated from the formulas 8-326, 8-327, values of *pBS* array and Table 8-15 of [JVTG050].

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

---

## FilterDeblockingLuma\_HorEdge\_H264

*Performs deblocking filtering on the horizontal edges of the 16X16 luma macroblock.*

---

### Syntax

```
IppStatus ippiFilterDeblockingLuma_HorEdge_H264_8u_C1IR(Ipp8u* pSrcDst,
    Ipp32s srcdstStep, Ipp8u* pAlpha, Ipp8u* pBeta, Ipp8u* pThresholds, Ipp8u*
    pBS);
```

### Parameters

<i>pSrcDst</i>	Pointer to the initial and resultant coefficients.
<i>srcdstStep</i>	Step of the array.
<i>pAlpha</i>	Array of size 2 of Alpha Thresholds (values for external and internal horizontal edge).
<i>pBeta</i>	Array of size 2 of Beta Thresholds (values for external and internal horizontal edge).
<i>pThresholds</i>	Array of size 16 of Thresholds ( $T_{c0}$ ) (values for upper edge of each 4x4 block).

*pBS*                      Array of size 16 of BS parameters (values for upper edge of each 4x4 block).

### Description

The function `ippiFilterDeblockingLuma_HorEdge_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs Deblocking Filtering on the horizontal edges of the 16x16 luma macroblock in accordance with 8.7.2 of [JVTG050].

The function uses arrays *pAlpha*, *pBeta*, *pBS*, *pThresholds* as input arguments, where *pAlpha*[0], *pBeta*[0] are values for the external horizontal edge, and *pAlpha*[1], *pBeta*[1] are values for the internal horizontal edge. See Figure 16-66 for the arrangement of *pThresholds* and *pBS* array elements.

Values of the arrays are calculated as follows:

- *pBS* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.
- *pAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].
- *pBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].
- *pThresholds*[*i*] values are calculated from the formulas 8-326, 8-327, values of *pBS* array and Table 8-15 of [JVTG050].

**Figure 16-66**    Arrangement of *pThresholds* Array Elements into a macroblock

---

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

This function is used in the H.264 decoder and encoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

---

## FilterDeblockingChroma\_VerEdge\_H264

*Performs deblocking filtering on the vertical edges of the 8X8 chroma macroblock.*

---

### Syntax

```
IppStatus ippiFilterDeblockingChroma_VerEdge_H264_8u_C1IR(Ipp8u* pSrcDst,  
    Ipp32s srcdstStep, Ipp8u* pAlpha, Ipp8u* pBeta, Ipp8u* pThresholds, Ipp8u*  
    pBS);
```

### Parameters

<code>pSrcDst</code>	Pointer to the initial and resultant coefficients.
<code>srcdstStep</code>	Step of the array.
<code>pAlpha</code>	Array of size 2 of Alpha Thresholds (values for external and internal vertical edge).
<code>pBeta</code>	Array of size 2 of Beta Thresholds (values for external and internal vertical edge).
<code>pThresholds</code>	Array of size 8 of Thresholds ( $T_{c0}$ ) (values for the left edge of each 2x2 block).
<code>pBS</code>	Array of size 16 of BS parameters (values for the left edge of each 2x2 block).

## Description

The function `ippiFilterDeblockingChroma_VerEdge_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs Deblocking Filtering on the vertical edge of the 8x8 chroma macroblock in accordance with 8.7.2 of [JVTG050].

The function uses arrays `pAlpha`, `pBeta`, `pBS`, `pThresholds` as input arguments. `pAlpha`, `pBeta`, and `pBS` are the same arrays as in [FilterDeblockingLuma\\_VerEdge\\_H264](#) function. `pAlpha[0]`, `pBeta[0]` are values for the external vertical edge, and `pAlpha[1]`, `pBeta[1]` are values for the internal vertical edge. See [Figure 16-67](#) for the arrangement of `pBS` array elements and [Figure 16-68](#) for the arrangement of `pThresholds` array elements.

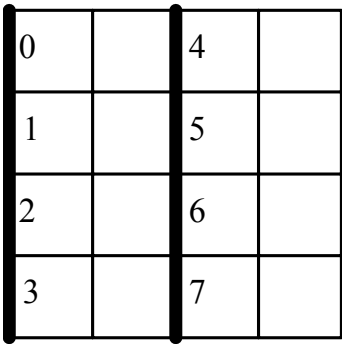
**Figure 16-67 Arrangement of `pBS` Array Elements into an 8x8 Chroma Block**

---

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

---

**Figure 16-68** Arrangement of *pThresholds* Array Elements into an 8x8 Chroma Block



As two vertical edges are filtered for the chroma component, the function uses *pBS* array elements with indices [0,3] for the external edge and [8,11] for the internal edge.

Values of the arrays are calculated as follows:

- *pBS* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.
- *pAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].
- *pBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].
- *pThresholds*[*i*] values are calculated from the formulas 8-326, 8-327, values of *pBS* array and Table 8-15 of [JVTG050].

This function is used in the H.264 decoder and encoder included into IPP Samples. See [introduction](#) to this section.

**Return Values**

- |                               |   |
|-------------------------------|---|
| <code>ippStsNoErr</code>      | Indicates no error.   |
| <code>ippStsNullPtrErr</code> | Indicates an error condition if at least one of the specified pointers is NULL. |

---

## FilterDeblockingChroma\_VerEdge\_MBAFF\_H264

*Performs deblocking filtering on the vertical edges of half of 8X8 chroma macroblock.*

---

### Syntax

```
IppStatus ippiFilterDeblockingChroma_VerEdge_MBAFF_H264_8u_C1IR( Ipp8u*  
    pSrcDst, Ipp32s srcdstStep, Ipp32u nAlpha, Ipp32u nBeta, Ipp8u* pThresholds,  
    Ipp8u* pBS );
```

### Parameters

<i>pSrcDst</i>	Pointer to the initial and resultant coefficients.
<i>srcdstStep</i>	Step of the array.
<i>pAlpha</i>	Alpha Threshold.
<i>pBeta</i>	Beta Threshold.
<i>pThresholds</i>	Array of size 4 of Thresholds ( $T_{c0}$ ) (two values for the left edge of each 2x2 block).
<i>pBS</i>	Array of size 4 of BS parameters (two values for the left edge of each 2x2 block).

### Description

The function `ippiFilterDeblockingChroma_VerEdge_MBAFF_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs Deblocking Filtering on the vertical edge of the 8x8 chroma macroblock in accordance with 8.7.2 of [JVTG050].

In process of filling *pThresholds* and *pBS* parameters for deblocking of each 2x2 block, a number of parameters of the neighboring blocks are taken into account. In MBAFF mode, the edge blocks may border on two others when the left and the current macroblocks are coded differently: one as a field, the other as a frame, and vice versa. As a result, two values of *pThresholds* and *pBS* parameters are taken for each block. For higher flexibility, the function processes only half of the macroblock to fit both types of the macroblocks, that is, coded as a field and coded as a frame.

The function uses arrays *nAlpha*, *nBeta*, *pBS*, *pThresholds* as input arguments. *nAlpha*, *nBeta*, and *pBS* are the same arrays as in [FilterDeblockingLuma\\_VerEdge\\_H264](#) function. See [Figure 16-67](#) for the arrangement of *pBS* array elements and [Figure 16-68](#) for the arrangement of *pThresholds* array elements.



Values of the arrays are calculated as follows:

- *pBS* values are calculated as per 8.7.2.1 of [JVTG050] and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.
- *nAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [JVTG050].
- *nBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [JVTG050].
- *pThresholds[i]* values are calculated from the formulas 8-326, 8-327, values of *pBS* array and Table 8-15 of [JVTG050].

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

---

## FilterDeblockingChroma\_HorEdge\_H264

*Performs deblocking filtering on the horizontal edges of the 8X8 chroma macroblock.*

---

### Syntax

```
IppStatus ippiFilterDeblockingChroma_HorEdge_H264_8u_C1IR(Ipp8u* pSrcDst,
    Ipp32s srcdstStep, Ipp8u* pAlpha, Ipp8u* pBeta, Ipp8u* pThresholds, ipp8u*
    pBS);
```

### Parameters

<i>pSrcDst</i>	Pointer to the initial and resultant coefficients.
<i>srcdstStep</i>	Step of the array.
<i>pAlpha</i>	Array of size 2 of Alpha Thresholds (values for external and internal horizontal edge).
<i>pBeta</i>	Array of size 2 of Beta Thresholds (values for external and internal horizontal edge).

<i>pThresholds</i>	Array of size 8 of Thresholds ( $T_{c0}$ ) (values for upper edge of each 2x2 block).
<i>pBS</i>	Array of size 16 of BS parameters (values for upper edge of each 2x2 block).

### Description

The function `ippiFilterDeblockingChroma_HorEdge_H264_8u_C1IR` is declared in the `ippvc.h` file. This function performs Deblocking Filtering on the horizontal edge of the 8x8 chroma macroblock in accordance with 8.7.2 of [JVTG050].

The function uses arrays *pAlpha*, *pBeta*, *pBS*, *pThresholds* as input arguments. *pAlpha*, *pBeta*, and *pBS* are the same arrays as in [FilterDeblockingLuma\\_HorEdge\\_H264](#) function. *pAlpha*[0], *pBeta*[0] are values for the external horizontal edge, and *pAlpha*[1], *pBeta*[1] are values for the internal horizontal edge. See [Figure 16-69](#) for the arrangement of *pBS* array elements and [Figure 16-70](#) for the arrangement of *pThresholds* array elements.

**Figure 16-69 Arrangement of *pBS* Array Elements into an 8x8 Chroma Block**

---

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

---

**Figure 16-70    Arrangement of *pThresholds* Array Elements into an 8x8 Chroma Block**

0	1	2	3
4	5	6	7

As two horizontal edges are filtered for the chroma component, the function uses *pBS* array elements with indices [0,3] for the external edge and [8,11] for the internal edge.

Values of the arrays are calculated as follows:

- *pBS* values are calculated as per 8.7.2.1 of [\[JVTG050\]](#) and may take the following values: 0 - if no edge is filtered; [1,3] - if filtering is weak; 4 - if filtering is strong.
  - *pAlpha* values are calculated from the formulas 8-326, 8-327 and Table 8-14 of [\[JVTG050\]](#).
  - *pBeta* values are calculated from the formulas 8-326, 8-328 and Table 8-14 of [\[JVTG050\]](#).
- pThresholds*[*i*] values are calculated from the formulas 8-326, 8-327, values of *pBS* array and Table 8-15 of [\[JVTG050\]](#).

This function is used in the H.264 decoder and encoder included into IPP Samples. See [introduction](#) to this section.

**Return Values**

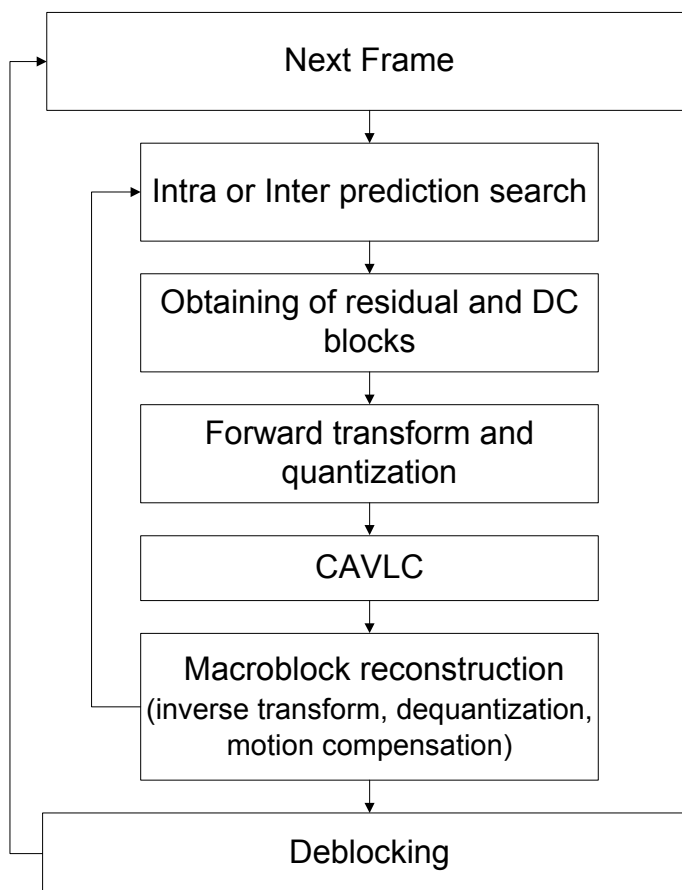
- |                               |   |
|-------------------------------|---|
| <code>ippStsNoErr</code>      | Indicates no error.   |
| <code>ippStsNullPtrErr</code> | Indicates an error condition if at least one of the specified pointers is NULL. |

## H.264 Encoder Functions

This section describes the functions for H.264 Encoder in accordance with JVT-G050 ([JVTG050]) standard.

**Figure 16-71 H.264 Encoder Structure**

---



Edges detection function ([EdgesDetect16x16](#)) is taken from [General Functions](#) category. This function helps to choose a mode of Intra macroblock coding: INTRA 16X16 or INTRA 4X4. If edges are detected inside the macroblock, both modes should be estimated. Otherwise, INTRA 16X16 mode should be selected.

Other general functions used by H.264 Encoder for predicted blocks estimation and for obtaining residual and DC blocks are [SAD Functions](#), [Sum of Differences Evaluation](#) functions, and [GetDiff4x4](#) function.

H.264 Encoder uses H.264 Decoder functions for calculation of [inter](#) and [intra predicted blocks](#), [Macroblock Reconstruction](#), and [Deblocking Filtering](#). Forward transform and quantization, and CAVLC coding are performed by Encoder proper functions, which are described below.

### Forward Transform and Quantization

---

## TransformQuantChromaDC\_H264

*Performs forward transform and quantization for 2X2 DC Chroma blocks.*

---

### Syntax

```
IppStatus ippiTransformQuantChromaDC_H264_16s_C1I(Ipp16s* pSrcDst, Ipp16s*
    pTBlock, Ipp32s QPChroma, Ipp8s* NumLevels, Ipp8u Intra, Ipp8u
    NeedTransform);
```

### Parameters

<i>pSrcDst</i>	Pointer to a 2x2 chroma DC block - source and destination array of size 4.
<i>pTBlock</i>	Pointer to a 2x2 transformed chroma DC block - source or destination array of size 4.
<i>QPChroma</i>	Quantization parameter for chroma, in range [0, 39].
<i>NumLevels</i>	Pointer to a value taht contains: <ul style="list-style-type: none"> <li>a negative value of a number of non-zero elements in block after quantization (when the first quantized element in block is not equal to zero),</li> <li>a number of non-zero elements in block after quantization (when the first quantized element in block is equal to zero).</li> </ul>

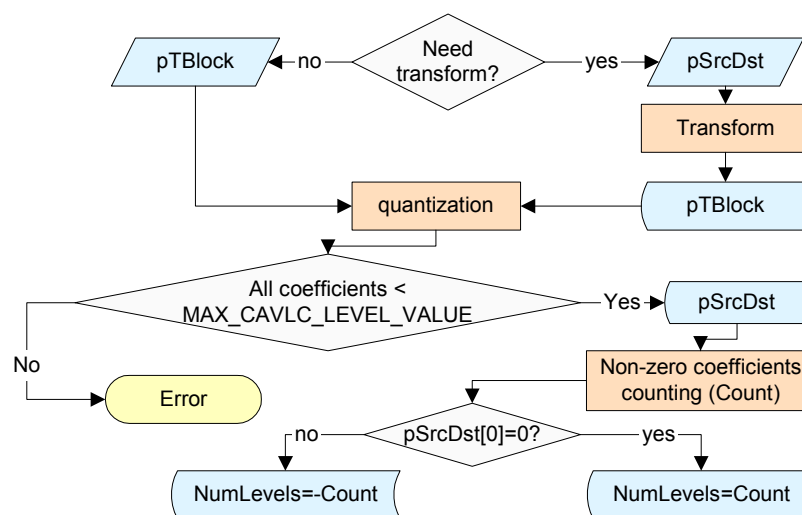
This value is calculated by the function.

<i>Intra</i>	Flag equal to 1 in the case of Intra macroblock, 0 otherwise.
<i>NeedTransform</i>	Flag that is equal to 1 if transforming process is used. This flag is equal to 0 if transforming process is not used.

## Description

The function `ippiTransformQuantChromaDC_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs forward transform, if necessary, and quantization for a 2x2 DC Chroma block as shown in [Figure 16-72](#) :

**Figure 16-72 Forward Transform and Quanization for 2x2 Block**



If *NeedTransform* is equal to 0, transformed DC block coefficients (*pTBlock*) are used for quantization.

If *NeedTransform* is equal to 1, at first a 2x2 chroma DC block *pSrcDst* is transformed and is saved in *pTBlock*. Then quatization process is performed on the transformed chroma DC block.

If all coefficients after quantization are not greater than `MAX_CAVLC_LEVEL_VALUE`, they are saved in *pSrcDst*. Otherwise, the function returns error.

This function is used in the H.264 encoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error if $QP_{Chroma} > 39$ or $QP_{Chroma} < 0$ .
<code>ippStsScaleRangeErr</code>	Indicates an error condition if any coefficient after quantization is greater than <code>MAX_CAVLC_LEVEL_VALUE</code> .

---

## TransformQuantLumaDC\_H264

*Performs forward transform and quantization for 4X4 DC Luma blocks.*

---

### Syntax

```
IppStatus ippTransformQuantLumaDC_H264_16s_C1I(Ipp16s* pSrcDst, Ipp16s*
    pTBlock, Ipp32s QP, Ipp8s* NumLevels, Ipp8u NeedTransform, Ipp16s*
    pScanMatrix, Ipp8u* LastCoeff);
```

### Parameters

<code>pSrcDst</code>	Pointer to a 4x4 luma DC block - source and destination array of size 16.
<code>pTBlock</code>	Pointer to a 4x4 transformed luma DC block - source or destination array of size 16.
<code>QP</code>	Quantization parameter for luma, in range [0, 51].
<code>NumLevels</code>	<p>Pointer to a value that contains:</p> <ul style="list-style-type: none"> <li>a negative value of a number of non-zero elements in block after quantization (when the first quantized element in block is not equal to zero),</li> <li>a number of non-zero elements in block after quantization (when the first quantized element in block is equal to zero).</li> </ul> <p>This value is calculated by the function.</p>
<code>NeedTransform</code>	Flag that is equal to 1 if transforming process is used. This flag is equal to 0 if transforming process is not used.

<i>pScanMatrix</i>	Scan matrix for coefficients in block (array of size 16).
<i>LastCoeff</i>	Position of the last (in order of <i>pScanMatrix</i> ) non-zero coefficient in block after quantization. This value is calculated by the function.

## Description

The function `ippiTransformQuantLumaDC_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs forward transform, if necessary, and quantization for a 4x4 DC Luma block as shown in [Figure 16-72](#) :

If *NeedTransform* is equal to 0, transformed DC block coefficients (*pTBlock*) are used for quantization.

If *NeedTransform* is equal to 1, at first a 4x4 Luma DC block *pSrcDst* is transformed and is saved in *pTBlock*. Then quatization process is performed on the transformed luma DC block.

If all coefficients after quantization are not greater than `MAX_CAVLC_LEVEL_VALUE`, they are saved in *pSrcDst*. Otherwise, the function returns error.

This function is used in the H.264 encoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error if $QP > 51$ or $QP < 0$ .
<code>ippStsScaleRangeErr</code>	Indicates an error condition if any coefficient after quantization is greater than <code>MAX_CAVLC_LEVEL_VALUE</code> .



## TransformQuantResidual\_H264

*Performs forward transform and quantization for 4X4 residual blocks.*

### Syntax

```
IppStatus ippiTransformQuantResidual_H264_16s_C1I(Ipp16s* pSrcDst, Ipp32s QP,
    Ipp8s* NumLevels, Ipp8u Intra, Ipp16s* pScanMatrix, Ipp8u* LastCoeff);
```

### Parameters

<i>pSrcDst</i>	Pointer to a 4x4 residual block - source and destination array of size 16.
<i>QP</i>	Quantization parameter for luma or for chroma, in range [0, 51] or [0, 39].
<i>NumLevels</i>	Pointer to a value that contains: <ul style="list-style-type: none"> <li>a negative value of a number of non-zero elements in block after quantization (when the first quantized element in block is not equal to zero),</li> <li>a number of non-zero elements in block after quantization (when the first quantized element in block is equal to zero).</li> </ul> This value is calculated by the function.
<i>Intra</i>	Flag equal to 1 in the case of Intra slice, 0 otherwise.
<i>pScanMatrix</i>	Scan matrix for coefficients in block (array of size 16).
<i>LastCoeff</i>	Position of the last (in order of <i>pScanMatrix</i> ) non-zero coefficient in block after quantization. This value is calculated by the function.

### Description

The function `ippiTransformQuantResidual_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs forward transform and quantization for a 4x4 residual block. :

A 4x4 residual block (*pSrcDst*) is transformed. Then quatization process is performed on the transformed residual block.

If all coefficients after quantization are not greater than `MAX_CAVLC_LEVEL_VALUE`, they are saved in *pSrcDst*. Otherwise, the function returns error.

This function is used in the H.264 encoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsOutOfRangeErr</code>	Indicates an error if $QP > 51$ or $QP < 0$ .
<code>ippStsScaleRangeErr</code>	Indicates an error condition if any coefficient after quantization is greater than <code>MAX_CAVLC_LEVEL_VALUE</code> .

---

## TransformLuma8x8Fwd\_H264

*Performs forward 8X8 transform for 8X8 Luma blocks without normalisation.*

---

### Syntax

```
IppStatus ippTransformLuma8x8Fwd_H264_16s_C1I(Ipp16s* pSrcDst);
```

### Parameters

`pSrcDst` Pointer to a 8x8 Luma block - source and destination array of size 64.

### Description

The function `ippTransformLuma8x8Fwd_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs forward transform for a 8x8 Luma block without final normalization (see `normAdjust8x8` matrix described in [\[ITUH264\]](#) :

### Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if the specified pointer is NULL.

---

## QuantLuma8x8\_H264

*Performs quantization for 8X8 Luma block coefficients including 8X8 transform normalization.*

---

### Syntax

```
IppStatus ippiQuantLuma8x8_H264_16s_C1(const Ipp16s *pSrc, Ipp16s *pDst, int Qp6, int Intra, const Ipp16s *pScanMatrix, const Ipp16s *pScaleLevels, int *pNumLevels, int *pLastCoeff);
```

### Parameters

<i>pSrc</i>	Pointer to source Luma block coefficients - array of size 64.
<i>pDst</i>	Pointer to the destination quantized block - array of size 64.
<i>QP6</i>	Quantization parameter divided by 6.
<i>Intra</i>	Flag that is equal to 1 if the slice is intra and 0 otherwise.
<i>pScanMatrix</i>	Pointer to a scan matrix for the coefficients in the block (array of size 64).
<i>pScaleLevels</i>	Pointer to a scale level matrix.
<i>pNumLevels</i>	Pointer to a value that contains: <ul style="list-style-type: none"><li>• a negative value of a number of non-zero elements in block after quantization (when the first quantized element in block is not equal to zero),</li><li>• a number of non-zero elements in block after quantization (when the first quantized element in block is equal to zero).</li></ul>
<i>pLastCoeff</i>	Position of the last (in order of <i>pScanMatrix</i> ) non-zero coefficient in block after quantization. This value is calculated by the function.

### Description

The function `ippiQuantLuma8x8_H264_16s_C1` is declared in the `ippvc.h` file. This function performs quantization of the coefficients of a Luma block after 8x8 transform including transform normalization (see `normAdjust8x8` matrix described in [\[ITUH264\]](#)).

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsQPErr</code>	Indicates an error if $Qp6 > 8$ or $Qp6 < 0$ .

---

## GenScaleLevel8x8\_H264

*Generates ScaleLevel matrices for forward and inverse quantization including normalization for 8X8 forward and inverse transform.*

---

## Syntax

```
IppStatus ippGenScaleLevel8x8_H264_8u16s_D2(const Ipp8u *pSrcInvScaleMatrix,
int SrcStep, Ipp16s *pDstInvScaleMatrix, Ipp16s *pDstScaleMatrix, int
QpRem);
```

## Parameters

<code>pSrcInvScaleMatrix</code>	Pointer to the source inverse scaling matrix for 8x8 transform.
<code>SrcStep</code>	Step of <code>pSrcInvScaleMatrix</code> , in bytes.
<code>pDstInvScaleMatrix</code>	Pointer to the destination inverse scaling matrix - array of size 64.
<code>pDstScaleMatrix</code>	Pointer to the destination forward scaling matrix - array of size 64.
<code>QpRem</code>	Reminder from integer division of the quantization parameter by 6.

## Description

The function `ippGenScaleLevel8x8_H264_8u16s_D2` is declared in the `ippvc.h` file. This function generates scaling matrices for forward and inverse quantization, taking into account normalization for forward and inverse 8x8 transform accordingly as defined.




---

**NOTE.** According to to [\[ITUH264\]](#) standard an original inverse scaling matrix `pSrcIncScaleMatrix` cannot contain zeroes.

---

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsQPErr</code>	Indicates an error if $QpRem > 5$ or $QpRem < 0$ .

## CAVLC Functions

These functions calculate block characteristics for CAVLC encoding:

- number of `trailing_ones` transform coefficient levels
- code that describes signs of `trailing_ones`
- number of non-zero coefficients in block
- number of zero coefficients in block
- array of levels
- array of runs.

`EncodeChromaDcCoeffsCAVLC_H264` is used for a 2x2 DC chroma block. For other block types `EncodeCoeffsCAVLC_H264` is used.

---

## EncodeCoeffsCAVLC\_H264

*Calculates characteristics of 4X4 block for CAVLC encoding.*

---

### Syntax

```
IppStatus ippEncodeCoeffsCAVLC_H264_16s(Ipp16s* pSrc, Ipp8u AC, Ipp32u
    *pScanMatrix, Ipp8u Count, Ipp8u *Traling_One, Ipp8u *Traling_One_Signs,
    Ipp8u *NumOutCoeffs, Ipp8u *TotalZeros, Ipp16s *pLevels, Ipp8u *pRuns);
```

### Parameters

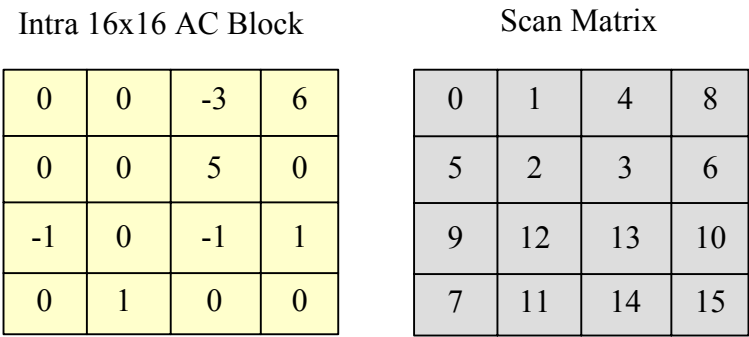
`pSrc`                      Pointer to 4x4 block - array of size 16.

<i>AC</i>	Flag that is equal to zero in the cases when zero coefficient should be encoded, and is equal to one otherwise.
<i>pScanMatrix</i>	Scan matrix for the coefficients in block (array of size 16).
<i>Count</i>	Position of the last non-zero block coefficient in the order of scan matrix. It should be in range [ <i>AC</i> ; 15].
<i>Trailing_One</i>	The number of “trailing ones” transform coefficient levels in a range [0;3]. This value is calculated by the function.
<i>Trailing_One_Signs</i>	Code that describes signs of “trailing ones”. ( <i>Trailing_One</i> - 1 - <i>i</i> )-bit in this code corresponds to a sign of <i>i</i> -“trailing one” in the current block. In this code 1 indicates negative value, 0 – positive value. This value is calculated by the function.
<i>NumOutCoeffs</i>	The number of non-zero coefficients in block (including “trailing ones”). This value is calculated by the function.
<i>TotalZeros</i>	The number of zero coefficients in block (except “trailing zeros”). This value is calculated by the function.
<i>pLevels</i>	Pointer to an array of size 16 that contains non-zero quantized coefficients of the current block (except “trailing ones”) in reverse scan matrix order. Elements of this array are calculated by the function.
<i>pRuns</i>	Pointer to an array of size 16 that contains runs before non-zero quantized coefficients (including “trailing ones”) of the current block in reverse scan matrix order (except run before the first non-zero coefficient in block, which can be calculated using <i>TotalZeros</i> ). Elements of this array are calculated by the function.

Description

The function `ippiEncodeCoeffsCAVLC_H264_16s` is declared in the `ippvc.h` file. This function calculates some characteristics (*Trailing\_One*, *Trailing\_One\_Signs*, *NumOutCoeffs*, *TotalZeros*, *pLevels*, *pRuns*) of a 4x4 block for CAVLC encoding. See [Figure 16-73](#) for an example of the function operation:

Figure 16-73



Count=13.

Figure 16-74 Arrangement of Block Coefficients After Scanning

coef	0	0	5	-3	0	0	0	6	-1	1	1	0	-1	0	0
pos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

*Trailing\_One* = 3 (-1, 1, 1)  
*Trailing\_One\_Signs* = 00000100  
*NumOutCoeffs* = 7  
*TotalZeros* = 6

**Figure 16-75 Array of Runs Before Non-Zero Coefficients (Except First) in Reverse Order**

---

pRuns	1	0	0	0	3	0
coefficient	-1	1	1	-1	6	-3

---

**Figure 16-76 Array of Non-Zero Coefficients (Except “Trailing Ones”) in Reverse Order**

---

pLevel	-1	6	-3	5
--------	----	---	----	---

---

This function is used in the H.264 encoder included into IPP Samples. See [introduction](#) to this section.

### Return Values

- `ippStsNoErr` Indicates no error.
- `ippStsNullPtrErr` Indicates an error condition if at least one of the specified pointers is NULL.
- `ippStsOutOfRangeErr` Indicates an error when *Count* is out of range [*AC*; 15].



---

## EncodeChromaDcCoeffsCAVLC\_H264

*Calculates characteristics of 2X2 Chroma DC block for CAVLC encoding.*

---

### Syntax

```
IppStatus ippiEncodeChromaDcCoeffsCAVLC_H264_16s(Ipp16s* pSrc, Ipp8u
    *Trailing_One, Ipp8u *Trailing_One_Signs, Ipp8u *NumOutCoeffs, Ipp8u
    *TotalZeros, Ipp16s *pLevels, Ipp8u *pRuns);
```

### Parameters

<i>pSrc</i>	Pointer to 2x2 block - array of size 4.
<i>Trailing_One</i>	The number of “trailing ones” transform coefficient levels in a range [0;3]. This value is calculated by the function.
<i>Trailing_One_Signs</i>	Code that describes signs of “trailing ones”. ( <i>Trailing_One</i> - 1 - <i>i</i> )-bit in this code corresponds to a sign of <i>i</i> -“trailing one” in the current block. In this code 1 indicates negative value, 0 – positive value. This value is calculated by the function.
<i>NumOutCoeffs</i>	The number of non-zero coefficients in block (including “trailing ones”). This value is calculated by the function.
<i>TotalZeros</i>	The number of zero coefficients in block (except “trailing zeros”). This value is calculated by the function.
<i>pLevels</i>	Pointer to an array of size 4 that contains non-zero quantized coefficients of the current block (except “trailing ones”) in reverse scan matrix order. Elements of this array are calculated by the function.
<i>pRuns</i>	Pointer to an array of size 4 that contains runs before non-zero quantized coefficients (including “trailing ones”) of the current block in reverse scan matrix order (except run before the first non-zero coefficient in block, which can be calculated using <i>TotalZeros</i> ). Elements of this array are calculated by the function.

## Description

The function `ippiEncodeChromaDcCoeffsCAVLC_H264_16s` is declared in the `ippvc.h` file. This function calculates some characteristics (*Trailing\_One*, *Trailing\_One\_Signs*, *NumOutCoeffs*, *TotalZeros*, *pLevels*, *pRuns*) of a 2x2 Chroma DC block for CAVLC encoding.

This function is used in the H.264 encoder included into IPP Samples. See [introduction](#) to this section.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

## Inverse Quantization and Transform

---

## QuantLuma8x8Inv\_H264

*Performs inverse quantization for 8X8 Luma block coefficients including normalization of the subsequent inverse 8X8 transform.*

---

## Syntax

```
IppStatus ippiQuantLuma8x8Inv_H264_16s_C1I(Ipp16s *pSrcDst, int Qp6, const
Ipp16s *pInvLevelScale);
```

## Parameters

<code>pSrcDst</code>	Pointer to Luma block coefficients - source and destination array of size 64.
<code>Qp6</code>	Quantization parameter divided by 6.
<code>pInvLevelScale</code>	Pointer to an inverse scale level matrix.

## Description

The function `ippiQuantLuma8x8Inv_H264_16s_C1I` is declared in the `ippvc.h` file. This function performs inverse quantization and normalization for 8x8 inverse transform of Luma block coefficients according to 8.5.13 of [ITUH264].

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.
<code>ippStsQPErr</code>	Indicates an error condition if $Qp6$ is less than 0 or greater than 8.

---

## TransformLuma8x8InvAddPred\_H264

*Performs inverse 8X8 transform of 8X8 Luma block coefficients with subsequent intra prediction or motion compensation.*

---

## Syntax

```
IppStatus ippiTransformLuma8x8InvAddPred_H264_16s8u_C1R(const Ipp8u *pPred, int  
PredStep, Ipp16s *pSrc, Ipp8u *pDst, int DstStep);
```

## Parameters

<i>pPred</i>	Pointer to a reference 8x8 block, which is used either for intra prediction or motion compensation.
<i>PredStep</i>	Reference frame step in bytes.
<i>pSrc</i>	Pointer to 8x8 Luma block coefficients – source and buffer array of size 64.
<i>pDst</i>	Pointer to the destination 8x8 block.
<i>DstStep</i>	Destination frame step in bytes.

## Description

The function `ippiTransformLuma8x8InvAddPred_H264_16s8u_C1R` is declared in the `ippvc.h` file. This function performs inverse transform of a 8x8 Luma block described in 8.5.13 of [ITUH264], with subsequent intra prediction or motion compensation. Input coefficients are assumed to be inverse quantized and normalized. The `pSrc` array is used as a buffer during computations.

## Return Values

<code>ippStsNoErr</code>	Indicates no error.
<code>ippStsNullPtrErr</code>	Indicates an error condition if at least one of the specified pointers is NULL.

# Handling of Special Cases



Some mathematical functions implemented in IPP are not defined for all possible argument values. This appendix describes how the corresponding IPP image processing functions handle situations when their input arguments fall outside the range of function definition or may lead to ambiguously determined output results.

[Table A-1](#) below summarizes these special cases for different functions and lists result values together with status codes returned by the functions. The status codes ending with `Err` (except for the `ippStsNoErr` status) indicate an error. When an error occurs, the function execution is interrupted. All other status codes indicate that the input argument is outside the range, but the function execution is continued with the corresponding result value.

**Table A-1      Special Cases for IPP Image Processing Functions**

Function Base Name	Data Type	Case Description	Result Value	Status Code
<a href="#">ippiSqrt</a>	16s	Sqrt (x <0)	0	ippStsSqrtNegArg
	32f	Sqrt (x <0)	NAN_32F	ippStsSqrtNegArg
<a href="#">ippiDiv</a>	8u	Div (0/0)	0	ippStsDivByZero
		Div (x/0)	IPP_MAX_8U	ippStsDivByZero
	16s	Div (0/0)	0	ippStsDivByZero
		Div (x/0), x>0	IPP_MAX_16S	ippStsDivByZero
		Div (x/0), x<0	IPP_MIN_16S	ippStsDivByZero
	32f	Div (0/0)	NAN_32F	ippStsDivByZero
		Div (x/0), x>0	INF_32F	ippStsDivByZero
		Div (x/0), x<0	INF_NEG_32F	ippStsDivByZero

**Table A-1 Special Cases for IPP Image Processing Functions (continued)**

Function Base Name	Data Type	Case Description	Result Value	Status Code
<a href="#">ippiDiv</a>	16sc	Div (0/0)	0	ppStsDivByZero
		Div (x/0),	0	ippStsDivByZero
	32sc	Div (0/0)	0	ippStsDivByZero
		Div (x/0), x>0	IPP_MAX_32S	ippStsDivByZero
		Div (x/0), x<0	IPP_MIN_32S	ippStsDivByZero
	32fc	Div (0/0)	NAN_32F	ippStsDivByZero
		Div (x/0), x>0	INF_32F	ippStsDivByZero
		Div (x/0), x<0	INF_NEG_32F	ippStsDivByZero
<a href="#">ippiDivC</a>	all	Div(x/const), const=0	-	ippStsDivByZeroErr
<a href="#">ippiLn</a>	8u	Ln (0)	0	ippStsLnZeroArg
	16s	Ln (0)	IPP_MIN_16S	ippStsLnZeroArg
		Ln (x<0)	IPP_MIN_16S	ippStsLnNegArg
	32f	Ln (x<0)	NAN_32F	ippStsLnNegArg
		Ln(x<IPP_MINABS_32F)	INF_NEG_32F	ippStsLnZeroArg
<a href="#">ippiExp</a>	8u	overflow	IPP_MAX_8U	ippStsNoErr
	16s	overflow	IPP_MAX_16S	ippStsNoErr
	32f	overflow	INF_32F	ippStsNoErr

Here x denotes an input value. For the definition of the constants used, see [“Image Data Types and Ranges”](#) in chapter 2.

Note that flavors of the same math function operating on different data types may produce unlike results for the equal argument values. However, for a given function and a fixed data type, handling of special cases is the same for all function flavors that have different [descriptors](#) in their

names. For example, logarithm function `ippiLn` operating on `16s` data treats zero argument values in the same way for all its flavors `ippiLn_16s_C1RSfs`, `ippiLn_16s_C3RSfs`, `ippiLn_16s_C1IRSfs`, and `ippiLn_16s_C3IRSfs`.

# Interpolation in Image Geometric Transform Functions



This appendix describes the interpolation algorithms used in the geometric transformation functions of the Intel IPP. For more information about each of the geometric transform functions, see [Chapter 12](#), Image Geometric Transforms.

## Overview of Interpolation Modes

In geometric transformations, the grid of input image pixels is not necessarily mapped onto the grid of pixels in the output image. Therefore, to compute the pixel intensities in the output image, the geometric transform functions need to interpolate the intensity values of several input pixels that are mapped to a certain neighborhood of the output pixel.

Geometric transformations can use various interpolation algorithms. When calling the geometric transform functions of the Intel IPP, the application code specifies the interpolation mode (that is, the type of interpolation algorithm) by using the parameter *interpolation*. The library supports the following interpolation modes:

nearest neighbor interpolation (*interpolation* = `IPPI_INTER_NN`)

linear interpolation (*interpolation* = `IPPI_INTER_LINEAR`)

cubic interpolation (*interpolation* = `IPPI_INTER_CUBIC`)

supersampling (*interpolation* = `IPPI_INTER_SUPER`)

interpolation with Lanczos window function (*interpolation* = `IPPI_INTER_LANCZOS`)

For certain functions, the above interpolation algorithms can be combined with additional smoothing (antialiasing) of edges to which the original image's borders are transformed. To use this edge smoothing, set the parameter *interpolation* to the bitwise OR of



IPPI\_SMOOTH\_EDGE and the desired interpolation mode. For example, in order to rotate an image with cubic interpolation and smooth the rotated image's edges, pass to `ippiRotate()` the following value:

```
interpolation = IPPI_INTER_CUBIC | IPPI_SMOOTH_EDGE.
```

Interpolation with edge smoothing option can be used only in those geometric transform functions where this option is explicitly listed in the arguments definition section.

[Table B-1](#) lists the supported interpolation modes for all geometric transform functions that use interpolation.

**Table B-1 Interpolation Modes Supported by Image Geometric Transform Functions**

Function Base Name	Nearest neighbor	Linear	Cubic	Super-sampling	Lanczos	Edge Smoothing
<a href="#">Resize</a>	X	X	X	X	X	
<a href="#">ResizeCenter</a>	X	X	X	X	X	
<a href="#">GetResizeFract</a>	X	X	X		X	
<a href="#">ResizeShift</a>	X	X	X		X	
<a href="#">Remap</a>	X	X	X			
<a href="#">Rotate</a>	X	X	X			X
<a href="#">RotateCenter</a>	X	X	X			X
<a href="#">Shear</a>	X	X	X			X
<a href="#">WarpAffine</a>	X	X	X			X
<a href="#">WarpAffineBack</a>	X	X	X			
<a href="#">WarpAffineQuad</a>	X	X	X			X
<a href="#">WarpPerspective</a>	X	X	X			X
<a href="#">WarpPerspectiveBack</a>	X	X	X			
<a href="#">WarpPerspectiveQuad</a>	X	X	X			
<a href="#">WarpBilinear</a>	X	X	X			X
<a href="#">WarpBilinearBack</a>	X	X	X			
<a href="#">WarpBilinearQuad</a>	X	X	X			X

The sections that follow provide more details on each interpolation mode.

## Mathematical Notation

In this appendix the following notation is used:

$(x_D, y_D)$	pixel coordinates in the destination image (integer values);
$(x_S, y_S)$	the computed coordinates of a point in the source image that is mapped exactly to $(x_D, y_D)$ ;
$S(x, y)$	pixel value (intensity) in the source image;
$D(x, y)$	pixel value (intensity) in the destination image.

## Nearest Neighbor Interpolation

This is the fastest and least accurate interpolation mode. The pixel value in the destination image is set to the value of the source image's pixel closest to the point  $(x_S, y_S)$ :  $D(x_D, y_D) = S(\text{round}(x_S), \text{round}(y_S))$ .

To use the nearest neighbor interpolation, set the parameter *interpolation* to `IPPI_INTER_NN`.

## Linear Interpolation

The linear interpolation is slower but more accurate than the nearest neighbor interpolation. On the other hand, it is faster but less accurate than cubic interpolation. The linear interpolation algorithm uses source image intensities at the four pixels

$(x_{S0}, y_{S0})$ ,  $(x_{SI}, y_{S0})$ ,  $(x_{S0}, y_{SI})$ ,  $(x_{SI}, y_{SI})$  which are closest to  $(x_S, y_S)$  in the source image:  
 $x_{S0} = \text{int}(x_S)$ ,  $x_{SI} = x_{S0} + 1$ ,  $y_{S0} = \text{int}(y_S)$ ,  $y_{SI} = y_{S0} + 1$ .

First, the intensity values are interpolated along the x-axis to produce two intermediate results  $I_0$  and  $I_I$  (see [Figure B-1](#)):

$$I_0 = S(x_S, y_{S0}) = S(x_{S0}, y_{S0}) * (x_{SI} - x_S) + S(x_{SI}, y_{S0}) * (x_S - x_{S0})$$

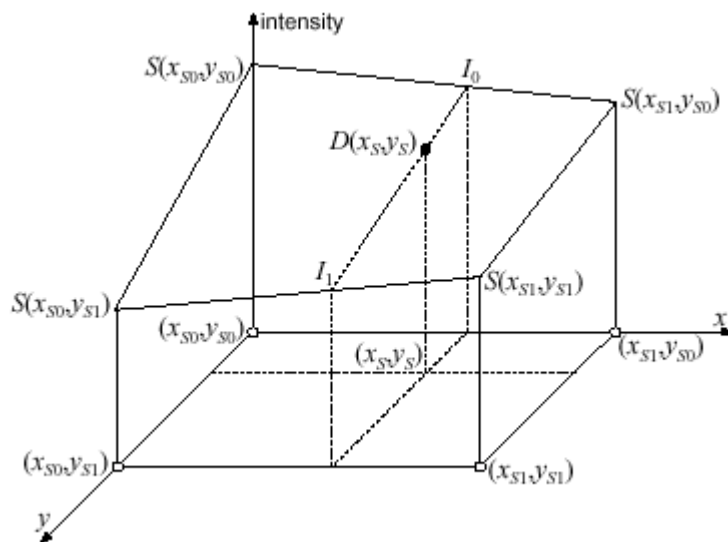
$$I_I = S(x_S, y_{SI}) = S(x_{S0}, y_{SI}) * (x_{SI} - x_S) + S(x_{SI}, y_{SI}) * (x_S - x_{S0}).$$

Then, the sought-for intensity  $D(x_D, y_D)$  is computed by interpolating the intermediate values  $I_0$  and  $I_I$  along the y-axis:

$$D(x_D, y_D) = I_0 * (y_{SI} - y_S) + I_I * (y_S - y_{S0}).$$

To use the linear interpolation, set the parameter *interpolation* to `IPPI_INTER_LINEAR`. For images with 8-bit unsigned color channels, the functions `ippiWarpAffine`, `ippiRotate`, and `ippiShear` compute the coordinates  $(x_S, y_S)$  with the accuracy  $2^{-16} = 1/65536$ . For images with 16-bit unsigned color channels, these functions compute the coordinates with floating-point precision.

**Figure B-1** Linear Interpolation



## Cubic Interpolation

The cubic interpolation algorithm (see [Figure B-2](#)) uses source image intensities at sixteen pixels in the neighborhood of the point  $(x_S, y_S)$  in the source image:

$$x_{S0} = \text{int}(x_S) - 1; \quad x_{S1} = x_{S0} + 1; \quad x_{S2} = x_{S0} + 2; \quad x_{S3} = x_{S0} + 3;$$

$$y_{S0} = \text{int}(y_S) - 1; \quad y_{S1} = y_{S0} + 1; \quad y_{S2} = y_{S0} + 2; \quad y_{S3} = y_{S0} + 3.$$

First, for each  $y_{Sk}$  the algorithm determines four cubic polynomials  $F_0(x)$ ,  $F_1(x)$ ,  $F_2(x)$ , and  $F_3(x)$ :

$$F_k(x) = a_k x^3 + b_k x^2 + c_k x + d_k, \quad 0 \leq k \leq 3$$

such that

$$\begin{aligned} F_k(x_{S0}) &= S(x_{S0}, y_{Sk}); & F_k(x_{S1}) &= S(x_{S1}, y_{Sk}), \\ F_k(x_{S2}) &= S(x_{S2}, y_{Sk}); & F_k(x_{S3}) &= S(x_{S3}, y_{Sk}). \end{aligned}$$

In [Figure B-2](#), these polynomials are shown by solid curves.

Then, the algorithm determines a cubic polynomial  $F_y(y)$  such that

$$F_y(y_{S0}) = F_0(x_S), \quad F_y(y_{S1}) = F_1(x_S), \quad F_y(y_{S2}) = F_2(x_S), \quad F_y(y_{S3}) = F_3(x_S).$$

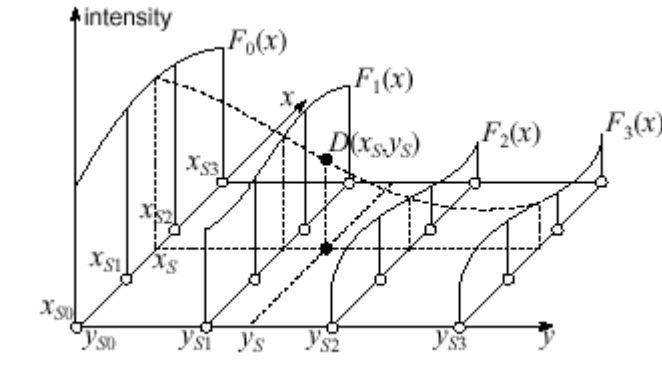
The polynomial  $F_y(y)$  is represented by the dashed curve in [Figure B-2](#).

Finally, the sought-for intensity  $D(x_D, y_D)$  is set to the value  $F_y(y_S)$ .

To use the cubic interpolation, set the parameter *interpolation* to `IPPI_INTER_CUBIC`.

For images with 8-bit unsigned color channels, the functions `ippiWarpAffine`, `ippiRotate`, and `ippiShear` compute the coordinates  $(x_S, y_S)$  with the accuracy  $2^{-16} = 1/65536$ . For images with 16-bit unsigned color channels, these functions compute the coordinates with floating-point precision.

**Figure B-2**    **Cubic Interpolation**



## Super Sampling

If the destination image is much smaller than the source image, the above interpolation algorithms may skip some pixels in the source image (that is, these algorithms not necessarily use all source pixels when computing the destination pixels' intensity). In order to use all pixel values of the source image, the `ippiResize` and `ippiResizeCenter` functions support the super-sampling algorithm, which is free of the above drawback.

The super-sampling algorithm is as follows:

1. Divide the source image's rectangular ROI (or the whole image, if there is no ROI) into equal rectangles, each rectangle corresponding to some pixel in the destination image. Note that each source pixel is represented by a 1x1 square.
2. Compute a weighted sum of source pixel values for all pixels that are in the rectangle or have a non-zero intersection with the rectangle. If a source pixel is fully contained in the rectangle, that pixel's value is taken with weight 1. If the rectangle and the source pixel's square have an intersection of area  $a < 1$ , that pixel's value is taken with weight  $a$ .

[Figure B-3](#) shows the corresponding weight value for each source pixel intersecting with the rectangle.

3. To compute the pixel value in the destination image, divide this weighted sum by the ratio of the source and destination rectangle areas  $S_{Src}/S_{Dst} = 1/xFactor*yFactor$ .

Here *xFactor*, and *yFactor* are the parameters of the functions that specify the factors by which the *x* and *y* dimensions of the source image ROI are changed.

Note that supersampling interpolation can be used only for  $xFactor < 1$ , and  $yFactor < 1$ .

**Figure B-3** Supersampling Weights

	$\Delta_2$	$\Delta_2$	$\Delta_2$	$\Delta_2 \times \Delta_3$
	1	1	1	$\Delta_3$
	1	1	1	$\Delta_3$
	$\Delta_1$	$\Delta_1$	$\Delta_1$	$\Delta_1 \times \Delta_3$

## Lanczos Interpolation

This method is based on the 3-lobed Lanczos window function as the interpolation function.

The interpolation algorithm uses source image intensities at 36 pixels in the neighborhood of the point  $(x_S, y_S)$  in the source image:

$$x_{S0} = \text{int}(x_S) - 2; \quad x_{SI} = x_{S0} + 1; \quad x_{S2} = x_{S0} + 2; \quad x_{S3} = x_{S0} + 3; \quad x_{S3} = x_{S0} + 4; \quad x_{S3} = x_{S0} + 5;$$

$$y_{S0} = \text{int}(y_S) - 2; \quad y_{SI} = y_{S0} + 1; \quad y_{S2} = y_{S0} + 2; \quad y_{S3} = y_{S0} + 3; \quad y_{S2} = y_{S0} + 4; \quad y_{S2} = y_{S0} + 5;$$

First, the intensity values are interpolated along the x-axis to produce six intermediate results  $I_0, I_1, \dots, I_5$ :

$$I_k = \sum_{i=0}^5 a_i \cdot S(x_{Si}, y_{Sk}) \quad 0 \leq k \leq 5.$$

Then the intensity  $D(x_D, y_D)$  is computed by interpolating the intermediate values  $I_k$  along the y-axis:

$$D(x_D, y_D) = \sum_{k=0}^5 b_k \cdot I_k$$

Here  $a_i$  and  $b_k$  are the coefficients defined as

$$a_i = L(x_S - x_{Si}), \quad b_k = L(y_S - y_{Sk}),$$

where  $L(x)$  is the Lanczos windowed sinc function:

$$L(x) = \text{sinc}(x) \cdot \text{Lanczos3}(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} \cdot \frac{\sin((\pi x)/3)}{(\pi x)/3} & 0 \leq |x| < 3 \\ 0 & 3 \leq |x| \end{cases}$$

To use this interpolation, set the parameter *interpolation* to `IPPI_INTER_LANCZOS`.

# Removed Functions



This appendix contains [Table C-1](#) that lists the functions that have been removed from the Intel IPP version 5.0. If an application created with the previous versions calls function listed here, then the source code must be modified. The table specifies the corresponding Intel IPP 5.0 functions to substitute for removed functions.

**Table C-1      Removed Functions**

Removed from 5.0	Substitution or Workaround	Comments
<b>Computer Vision Fucntions</b>		
ippiBlur_32f_C1R	ippiFilterLowpassBorder_32f_C1R	
ippiBlur_8s16s_C1R	ippiConvert_8s32f_C1R, ippiFilterBox_32f_C1R, ippiConvert_32f16s_C1R	
ippiBlur_8u16s_C1R	ippiFilterLowpassBorder_8u_C1R	see also ippiFilterBox
ippiBlurInitAlloc		no structure, useless
ippiConvFree		no structure, useless
ippiDilateStrip_32f_C1R	ippiDilateBorderReplicate_32f_C1R	
ippiDilateStrip_32f_C3R	ippiDilateBorderReplicate_32f_C3R	
ippiDilateStrip_32f_C4R	ippiDilateBorderReplicate_32f_C4R	
ippiDilateStrip_8u_C1R	ippiDilateBorderReplicate_8u_C1R	
ippiDilateStrip_8u_C3R	ippiDilateBorderReplicate_8u_C3R	
ippiDilateStrip_8u_C4R	ippiDilateBorderReplicate_8u_C4R	
ippiDilateStrip_Cross_32f_C1R	ippiDilateBorderReplicate_32f_C1R	
ippiDilateStrip_Cross_32f_C3R	ippiDilateBorderReplicate_32f_C3R	
ippiDilateStrip_Cross_32f_C4R	ippiDilateBorderReplicate_32f_C4R	

**Table C-1**      **Removed Functions** (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiDilateStrip_Cross_8u_C1R	ippiDilateBorderReplicate_8u_C1R	
ippiDilateStrip_Cross_8u_C3R	ippiDilateBorderReplicate_8u_C3R	
ippiDilateStrip_Cross_8u_C4R	ippiDilateBorderReplicate_8u_C4R	
ippiDilateStrip_Ellipse_32f_C1R	ippiDilateBorderReplicate_32f_C1R	
ippiDilateStrip_Ellipse_32f_C3R	ippiDilateBorderReplicate_32f_C3R	
ippiDilateStrip_Ellipse_32f_C4R	ippiDilateBorderReplicate_32f_C4R	
ippiDilateStrip_Ellipse_8u_C1R	ippiDilateBorderReplicate_8u_C1R	
ippiDilateStrip_Ellipse_8u_C3R	ippiDilateBorderReplicate_8u_C3R	
ippiDilateStrip_Ellipse_8u_C4R	ippiDilateBorderReplicate_8u_C4R	
ippiDilateStrip_Rect_32f_C1R	ippiDilateBorderReplicate_32f_C1R	
ippiDilateStrip_Rect_32f_C3R	ippiDilateBorderReplicate_32f_C3R	
ippiDilateStrip_Rect_32f_C4R	ippiDilateBorderReplicate_32f_C4R	
ippiDilateStrip_Rect_8u_C1R	ippiDilateBorderReplicate_8u_C1R	
ippiDilateStrip_Rect_8u_C3R	ippiDilateBorderReplicate_8u_C3R	
ippiDilateStrip_Rect_8u_C4R	ippiDilateBorderReplicate_8u_C4R	
ippiEigenValsVecs_8s32f_C1R		useless
ippiEigenValsVecsGetSize	ippiEigenValsVecsGetBufferSize_8u32f_C1R, ippiEigenValsVecsGetBufferSize_32f_C1R	
ippiErodeStrip_32f_C1R	ippiErodeBorderReplicate_32f_C1R	
ippiErodeStrip_32f_C3R	ippiErodeBorderReplicate_32f_C3R	
ippiErodeStrip_32f_C4R	ippiErodeBorderReplicate_32f_C4R	
ippiErodeStrip_8u_C1R	ippiErodeBorderReplicate_8u_C1R	
ippiErodeStrip_8u_C3R	ippiErodeBorderReplicate_8u_C3R	
ippiErodeStrip_8u_C4R	ippiErodeBorderReplicate_8u_C4R	
ippiErodeStrip_Cross_32f_C1R	ippiErodeBorderReplicate_32f_C1R	
ippiErodeStrip_Cross_32f_C3R	ippiErodeBorderReplicate_32f_C3R	
ippiErodeStrip_Cross_32f_C4R	ippiErodeBorderReplicate_32f_C4R	
ippiErodeStrip_Cross_8u_C1R	ippiErodeBorderReplicate_8u_C1R	
ippiErodeStrip_Cross_8u_C3R	ippiErodeBorderReplicate_8u_C3R	
ippiErodeStrip_Cross_8u_C4R	ippiErodeBorderReplicate_8u_C4R	



**Table C-1**      **Removed Functions** (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiErodeStrip_Ellipse_32f_C1R	ippiErodeBorderReplicate_32f_C1R	
ippiErodeStrip_Ellipse_32f_C3R	ippiErodeBorderReplicate_32f_C3R	
ippiErodeStrip_Ellipse_32f_C4R	ippiErodeBorderReplicate_32f_C4R	
ippiErodeStrip_Ellipse_8u_C1R	ippiErodeBorderReplicate_8u_C1R	
ippiErodeStrip_Ellipse_8u_C3R	ippiErodeBorderReplicate_8u_C3R	
ippiErodeStrip_Ellipse_8u_C4R	ippiErodeBorderReplicate_8u_C4R	
ippiErodeStrip_Rect_32f_C1R	ippiErodeBorderReplicate_32f_C1R	
ippiErodeStrip_Rect_32f_C3R	ippiErodeBorderReplicate_32f_C3R	
ippiErodeStrip_Rect_32f_C4R	ippiErodeBorderReplicate_32f_C4R	
ippiErodeStrip_Rect_8u_C1R	ippiErodeBorderReplicate_8u_C1R	
ippiErodeStrip_Rect_8u_C3R	ippiErodeBorderReplicate_8u_C3R	
ippiErodeStrip_Rect_8u_C4R	ippiErodeBorderReplicate_8u_C4R	
ippiLaplace_32f_C1R	ippiFilterLaplacianBorder_32f_C1R	
ippiLaplace_8s16s_C1R		
ippiLaplace_8u16s_C1R	ippiFilterLaplacianBorder_8u16s_C1R	
ippiLaplaceInitAlloc		no structure, useless
ippiMatchTemplate_Coeff_32f_C1R		useless
ippiMatchTemplate_Coeff_8s32f_C1R		useless
ippiMatchTemplate_Coeff_8u32f_C1R		useless
ippiMatchTemplate_CoeffNormed_32f_C1R	ippiCrossCorrValid_NormLevel_32f_C1R	
ippiMatchTemplate_CoeffNormed_8s32f_C1R	ippiCrossCorrValid_NormLevel_8s32f_C1R	
ippiMatchTemplate_CoeffNormed_8u32f_C1R	ippiCrossCorrValid_NormLevel_8u32f_C1R	
ippiMatchTemplate_Corr_32f_C1R		useless
ippiMatchTemplate_Corr_8s32f_C1R		useless
ippiMatchTemplate_Corr_8u32f_C1R		useless
ippiMatchTemplate_CorrNormed_32f_C1R	ippiCrossCorrValid_Norm_32f_C1R	
ippiMatchTemplate_CorrNormed_8s32f_C1R	ippiCrossCorrValid_Norm_8s32f_C1R	

**Table C-1**      **Removed Functions** (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiMatchTemplate_CorrNormed_8u32f_C1R	ippiCrossCorrValid_Norm_8u32f_C1R	
ippiMatchTemplate_SqDiff_32f_C1R		useless
ippiMatchTemplate_SqDiff_8s32f_C1R		useless
ippiMatchTemplate_SqDiff_8u32f_C1R		useless
ippiMatchTemplate_SqDiffNormed_32f_C1R	ippiSqrDistanceValid_Norm_32f_C1R	
ippiMatchTemplate_SqDiffNormed_8s32f_C1R	ippiSqrDistanceValid_Norm_8s32f_C1R	
ippiMatchTemplate_SqDiffNormed_8u32f_C1R	ippiSqrDistanceValid_Norm_8u32f_C1R	
ippiMatchTemplateGetBufSize_Coeff		useless
ippiMatchTemplateGetBufSize_CoeffNormed		useless
ippiMatchTemplateGetBufSize_Corr		useless
ippiMatchTemplateGetBufSize_CorrNormed		useless
ippiMatchTemplateGetBufSize_SqDiff		useless
ippiMatchTemplateGetBufSize_SqDiffNormed		useless
ippiMinEigenVal_8s32f_C1R		
ippiMinEigenValGetSize	ippiMinEigenValGetBufferSize_8u32f_C1R, or ippiMinEigenValGetBufferSize_32f_C1R	
ippiMorphologyInitAlloc		no structure, useless
ippiScharr_Dx_32f_C1R	ippiFilterScharrHorizBorder_32f_C1R	
ippiScharr_Dx_8s16s_C1R		
ippiScharr_Dx_8u16s_C1R	ippiFilterScharrHorizBorder_8u16s_C1R	
ippiScharr_Dy_32f_C1R	ippiFilterScharrVertBorder_32f_C1R	
ippiScharr_Dy_8s16s_C1R		
ippiScharr_Dy_8u16s_C1R	ippiFilterScharrVertBorder_8u16s_C1R	

**Table C-1**      **Removed Functions** (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiSobel_32f_C1R	ippiFilterSobelHorizBorder_32f_C1R ippiFilterSobelVertBorder_32f_C1R	
ippiSobel_8s16s_C1R		
ippiSobel_8u16s_C1R	ippiFilterSobelHorizBorder_8u16s_C1R, or ippiFilterSobelHorizBorder_8u16s_C1R	
ippiSobel3x3_D2x_32f_C1R	ippiFilterSobelHorizSecondBorder_32f_C1R	
ippiSobel3x3_D2x_8s16s_C1R		
ippiSobel3x3_D2x_8u16s_C1R	ippiFilterSobelHorizSecondBorder_8u16s_C1R	
ippiSobel3x3_D2y_32f_C1R	ippiFilterSobelVertSecondBorder_32f_C1R	
ippiSobel3x3_D2y_8s16s_C1R		
ippiSobel3x3_D2y_8u16s_C1R	ippiFilterSobelVertSecondBorder_8u16s_C1R	
ippiSobel3x3_Dx_32f_C1R	ippiFilterSobelHorizBorder_32f_C1R	
ippiSobel3x3_Dx_8s16s_C1R		
ippiSobel3x3_Dx_8u16s_C1R	ippiFilterSobelHorizBorder_8u16s_C1R	
ippiSobel3x3_Dy_32f_C1R	ippiFilterSobelVertBorder_32f_C1R	
ippiSobel3x3_Dy_8s16s_C1R		
ippiSobel3x3_Dy_8u16s_C1R	ippiFilterSobelVertBorder_8u16s_C1R	
ippiSobel3x3_DxDy_32f_C1R	ippiFilterSobelCrossBorder_32f_C1R	
ippiSobel3x3_DxDy_8s16s_C1R		
ippiSobel3x3_DxDy_8u16s_C1R	ippiFilterSobelCrossBorder_8u16s_C1R	
ippiSobelInitAlloc		no structure, useless
<b>Color Conversion Functions</b>		
ippiJoin420_8u_P2C2R	ippiYCbCr420ToYCbCr422_8u_P2C2R	renamed, old name is supported
ippiJoin420_Filter_8u_P2C2R	ippiYCbCr420ToYCbCr422_Filter_8u_P2C2R	renamed, old name is supported
ippiJoin422_8u_P3C2R	ippiYCbCr422_8u_P3C2R	renamed, old name is supported

**Table C-1**      **Removed Functions** (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiSplit420_8u_P2P3R	ippiYCbCr420ToYCrCb420_8u_P2P3R	renamed, old name is supported
ippiSplit420_Filter_8u_P2P3R	ippiYCbCr420ToYCrCb420_Filter_8u_P2P3R	renamed, old name is supported
ippiSplit422_8u_C2P3R	ippiYCbCr422_8u_C2P3R	renamed, old name is supported
ippiUYToYU422_8u_C2P2R	ippiCbYCr422ToYCbCr420_8u_C2P2R	renamed, old name is supported
ippiUYToYU422_8u_C2R	ippiCbYCr422ToYCbCr422_8u_C2R	renamed, old name is supported
ippiUYToYV422_8u_C2P3R	ippiCbYCr422ToYCrCb420_8u_C2P3R	renamed, old name is supported
ippiYCbCr411ToYCbCr411_8u_P2P3R	ippiYCbCr411_8u_P2P3R	renamed, old name is supported
ippiYCbCr411ToYCbCr411_8u_P3P2R	ippiYCbCr411_8u_P3P2R	renamed, old name is supported
ippiYCbCr420ToYCbCr420_8u_P2P3R	ippiYCbCr420_8u_P2P3R	renamed, old name is supported
ippiYCbCr420ToYCbCr420_8u_P3P2R	ippiYCbCr420_8u_P3P2R	renamed, old name is supported
ippiYCbCr420ToYCbCr422Filter_8u_P2P3R	ippiYCbCr420ToYCbCr422_Filter_8u_P2P3R	renamed, old name is supported
ippiYCbCr420ToYCbCr422Filter_8u_P3R	ippiYCbCr420ToYCbCr422_Filter_8u_P3R	renamed, old name is supported
ippiYCbCr422ToYCbCr422_8u_C2P3R	ippiYCbCr422_8u_C2P3R	renamed, old name is supported
ippiYCrCb420ToYCbCr422Filter_8u_P3R	ippiYCrCb420ToYCbCr422_Filter_8u_P3R	renamed, old name is supported
ippiYUToUY420_8u_P2C2R	ippiYCbCr420ToCbYCr422_8u_P2C2R	renamed, old name is supported
ippiYUToUY422_8u_C2R	ippiYCbCr422ToCbYCr422_8u_C2R	renamed, old name is supported
ippiYUToYU422_8u_C2P2R	ippiYCbCr422ToYCbCr420_8u_C2P2R	renamed, old name is supported
ippiYUToYV422_8u_C2P3R	ippiYCbCr422ToYCrCb420_8u_C2P3R	renamed, old name is supported
ippiYVToUY420_8u_P3C2R	ippiYCrCb420ToCbYCr422_8u_P3C2R	renamed, old name is supported

**Table C-1**      **Removed Functions** (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiYVToYU420_8u_P3C2R	ippiYCrCb420ToYCbCr422_8u_P3C2R	renamed, old name is supported
ippiYVToYU420_8u_P3P2R	ippiYCrCb420ToYCbCr420_8u_P3P2R	renamed, old name is supported
ippiYCbCr420ToYCbCr420_8u_P2P3R	ippiYCbCr420_8u_P2P3R	renamed, old name is supported
ippiYCbCr420ToYCbCr420_8u_P3P2R	ippiYCbCr420_8u_P3P2R	renamed, old name is supported
ippiYCbCr420ToYCbCr422Filter_8u_P2P3R	ippiYCbCr420ToYCbCr422_Filter_8u_P2P3R	renamed, old name is supported
ippiYCbCr420ToYCbCr422Filter_8u_P3R	ippiYCbCr420ToYCbCr422_Filter_8u_P3R	renamed, old name is supported
<b>Image Processing Functions</b>		
ippiDCTInv_8x8_16s8u	ippiDCT8x8Inv_16s8u_C1R	
ippiDrawText_8u_C3R		empty function
ippiMalloc_16s_P3	ippiMalloc_16s_C1	3 calls with len, or 1 call with 3*len
ippiMalloc_16u_P3	ippiMalloc_16u_C1	3 calls with len, or 1 call with 3*len
ippiMalloc_32f_P3	ippiMalloc_32f_C1	3 calls with len, or 1 call with 3*len
ippiMalloc_32fc_P3	ippiMalloc_32fc_C1	3 calls with len, or 1 call with 3*len
ippiMalloc_32s_P3	ippiMalloc_32s_C1	3 calls with len, or 1 call with 3*len
ippiMalloc_32sc_P3	ippiMalloc_32sc_C1	3 calls with len, or 1 call with 3*len
ippiMalloc_8u_P3	ippiMalloc_8u_C1	3 calls with len, or 1 call with 3*len
<b>Video Coding Functions</b>		
ippiAverageBlock_MPEG4_8u		useless
ippiAverageBlock_MPEG4_8u_I	ippiAverage8x8_8u_C1IR	
ippiAverageMB_MPEG4_8u		useless
ippiAverageMB_MPEG4_8u_I	ippiAverage16x16_8u_C1IR	

**Table C-1**      **Removed Functions** (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiBlockMatch_Integer_16x16_MVFAST		functionality on codec level
ippiBlockMatch_Integer_16x16_SEA		functionality on codec level
ippiComputeChroma4MV_MPEG4		functionality on codec level
ippiComputeChromaMV_MPEG4		functionality on codec level
ippiComputeTextureErrorBlock_8u16s	ippiSub8x8_8u16s_C1R	
ippiComputeTextureErrorBlock_SAD_8u16s	ippiSubSAD8x8_8u16s_C1R	
ippiCopyApproxHBlock_H263_8u	ippiCopy8x8HP_8u_C1R	
ippiCopyApproxHMB_H263_8u	ippiCopy16x16HP_8u_C1R	
ippiCopyApproxHVBBlock_H263_8u	ippiCopy8x8HP_8u_C1R	
ippiCopyApproxHVMB_H263_8u	ippiCopy16x16HP_8u_C1R	
ippiCopyApproxVBBlock_H263_8u	ippiCopy8x8HP_8u_C1R	
ippiCopyApproxVMB_H263_8u	ippiCopy16x16HP_8u_C1R	
ippiCopyBlock_16x16_8u	ippiCopy16x16_8u_C1R	
ippiCopyBlock_8x8_8u	ippiCopy8x8_8u_C1R	
ippiCopyBlock_H263_8u	ippiCopy8x8_8u_C1R	
ippiCopyBlockHalfpel_MPEG4_8u	ippiCopy8x8HP_8u_C1R	
ippiCopyMB_H263_8u	ippiCopy16x16_8u_C1R	
ippiCopyMBHalfpel_MPEG4_8u	ippiCopy16x16HP_8u_C1R	
ippiDecodeBlockCoef_AdvIntra_H263_1u8u	ippiDecodeCoefIntra_H263_1u16s; QuantInv; DCTInv	set of functions
ippiDecodeBlockCoef_Inter_H263_1u16s	ippiDecodeCoefInter_H263_1u16s; QuantInv; DCTInv	set of functions
ippiDecodeBlockCoef_Inter_MPEG4_1u16s	ippiDecodeCoefInter_MPEG4_1u16s; QuantInv; DCTInv	set of functions
ippiDecodeBlockCoef_Intra_H263_1u8u	ippiDecodeDCIntra_H263_1u16s; QuantInv; DCTInv	set of functions
ippiDecodeBlockCoef_Intra_MPEG4_1u8u	ippiDecodeDCIntra_MPEG4_1u16s; QuantInv; DCTInv	set of functions

**Table C-1**      **Removed Functions** (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiDecodeBlockCoef_IntraDCOnly_H263_1u8u	ippiDecodeCoefIntra_H263_1u16s; QuantInv; DCTInv	set of functions
ippiDecodeBlockCoef_IntraDCOnly_MPEG4_1u8u	ippiDecodeACIntra_MPEG4_1u16s; QuantInv; DCTInv	set of functions
ippiDecodeCAEInterH_MPEG4_1u8u		functionality on codec level
ippiDecodeCAEInterV_MPEG4_1u8u		functionality on codec level
ippiDecodeCAEIntraH_MPEG4_1u8u		functionality on codec level
ippiDecodeCAEIntraV_MPEG4_1u8u		functionality on codec level
ippiDecodeCBPY_H263		functionality on codec level
ippiDecodeMCBPCInter_H263		functionality on codec level
ippiDecodeMCBPCIntra_H263		functionality on codec level
ippiDecodeMODB_H263		functionality on codec level
ippiDecodeMV_BVOP_Backward_MPEG4		functionality on codec level
ippiDecodeMV_BVOP_Direct_MPEG4		functionality on codec level
ippiDecodeMV_BVOP_DirectSkip_MPEG4		functionality on codec level
ippiDecodeMV_BVOP_Forward_MPEG4		functionality on codec level
ippiDecodeMV_BVOP_Interpolate_MPEG4		functionality on codec level
ippiDecodeMV_H263		functionality on codec level
ippiDecodeMV_TopBorder_H263		functionality on codec level
ippiDecodeMVS_MPEG4		functionality on codec level
ippiDecodePadMV_PVOP_MPEG4		functionality on codec level
ippiDecodeVLC_IntraDCVLC_MPEG4_1u16s	ippiDecodeDCIntra_MPEG4_1u16s	
ippiDecodeVLCZigzag_Inter_MPEG4_1u16s	ippiDecodeCoefInter_MPEG4_1u16s	
ippiDecodeVLCZigzag_IntraACVLC_MPEG4_1u16s	ippiDecodeACIntra_MPEG4_1u16s	
ippiDecodeVLCZigzag_IntraDCVLC_MPEG4_1u16s	ippiDecodeDCIntra_MPEG4_1u16s	

**Table C-1**      **Removed Functions** (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiEncode_ACVLC_H263_16s1u	ippiEncodeCoeffIntra_H263_16s1u	
ippiEncode_InterVLC_MPEG4_16s1u	ippiEncodeCoefInter_MPEG4_1u16s	
ippiEncode_IntraACVLC_MPEG4_16s1u	ippiEncodeACIntra_MPEG4_1u16s	
ippiEncode_IntraDCVLC_H263_16s1u	ippiEncodeDCIntra_H263_16s1u	
ippiEncode_IntraDCVLC_MPEG4_16s1u	ippiEncodeDCIntra_MPEG4_1u16s	
ippiEncodeMV_MPEG4_8u16s		functionality on codec level
ippiEncodeVLCZigzag_Inter_MPEG4_16s1u	ippiEncodeCoefInter_MPEG4_1u16s	
ippiEncodeVLCZigzag_IntraACVLC_MPEG4_16s1u	ippiEncodeACIntra_MPEG4_1u16s	
ippiEncodeVLCZigzag_IntraDCVLC_MPEG4_16s1u	ippiEncodeACIntra_MPEG4_1u16s	
ippiExpandFrame_H263_8u		functionality on codec level
ippiFilterDeblocking_HorEdge_H263_8u_I	ippiFilterDeblockingHorEdge_H263_8u_C1IR	
ippiFilterDeblocking_HorEdge_MPEG4_8u_I	ippiFilterDeblockingHorEdge_MPEG4_8u_C1IR	
ippiFilterDeblocking_VerEdge_H263_8u_I	ippiFilterDeblockingVerEdge_H263_8u_C1IR	
ippiFilterDeblocking_VerEdge_MPEG4_8u_I	ippiFilterDeblockingVerEdge_MPEG4_8u_C1IR	
ippiFilterDeringingSmoothBlock_MPEG4_8u	ippiFilterDeringingSmoothBlock_MPEG4_8u_C1R	
ippiFilterDeringingThresholdMB_MPEG4_8u	ippiFilterDeringingThresholdMB_MPEG4_8u_P3R	
ippiFindMVPred_MPEG4		functionality on codec level
ippiLimitMVToRect_MPEG4		functionality on codec level
ippiMCBlock_RoundOff_8u	ippiCopy8x8HP_8u_C1R	
ippiMCBlock_RoundOn_8u	ippiCopy8x8HP_8u_C1R	
ippiMotionEstimation_16x16_MVFAST		functionality on codec level
ippiMotionEstimation_16x16_SEA		functionality on codec level
ippiOBMCHalfpel_MPEG4_8u	ippiOBMC8x8HP_MPEG4_8u_C1R	



**Table C-1      Removed Functions (continued)**

Removed from 5.0	Substitution or Workaround	Comments
ippiPadCurrent_16x16_MPEG4_8u_I		functionality on codec level
ippiPadCurrent_8x8_MPEG4_8u_I		functionality on codec level
ippiPadMBGray_MPEG4_8u		functionality on codec level
ippiPadMBHorizontal_MPEG4_8u		functionality on codec level
ippiPadMBOpaque_MPEG4_8u_P4R		functionality on codec level
ippiPadMBPartial_MPEG4_8u_P4R		functionality on codec level
ippiPadMBTransparent_MPEG4_8u_P4R		functionality on codec level
ippiPadMBVertical_MPEG4_8u		functionality on codec level
ippiPadMV_MPEG4		functionality on codec level
ippiPredictBlock_OBMC_8u	ippiOBMC8x8HP_MPEG4_8u_C1R	
ippiPredictReconCoefIntra_MPEG4_16s		functionality on codec level
ippiQuant_H263_C1I	ippiQuantInter_H263_16s_C1I	
ippiQuant_MPEG4_16s_C1I	ippiQuantInter_H263_16s_C1I	
ippiQuantInter_MPEG4_16s_I	ippiQuantInter_H263_16s_C1I	
ippiQuantIntra_H263_C1I	ippiQuantIntra_H263_16s_C1I	
ippiQuantIntra_MPEG4_16s_I	ippiQuantIntra_H263_16s_C1I	
ippiQuantInv_H263_C1I	ippiQuantInvInter_H263_16s_C1I	
ippiQuantInv_MPEG4_16s_C1I	ippiQuantInvInter_MPEG4_16s_C1I	
ippiQuantInvInter_Compact_H263_16s_I	ippiQuantInvInter_H263_16s_C1I	
ippiQuantInvInter_MPEG4_16s_I	ippiQuantInvInter_MPEG4_16s_C1I	
ippiQuantInvInterFirst_MPEG4_16s_I	ippiQuantInvInter_MPEG4_16s_C1I	
ippiQuantInvInterSecond_MPEG4_16s_I	ippiQuantInvInter_MPEG4_16s_C1I	
ippiQuantInvIntra_Compact_H263_16s_I	ippiQuantInvIntra_H263_16s_C1I	
ippiQuantInvIntra_H263_C1I	ippiQuantInvIntra_H263_16s_C1I	
ippiQuantInvIntra_MPEG4_16s_I	ippiQuantInvIntra_MPEG4_16s_C1I	
ippiQuantInvIntraFirst_MPEG4_16s_I	ippiQuantInvIntra_MPEG4_16s_C1I	
ippiQuantInvIntraSecond_MPEG4_16s_I	ippiQuantInvIntra_MPEG4_16s_C1I	

**Table C-1**      **Removed Functions** (continued)

Removed from 5.0	Substitution or Workaround	Comments
ippiReconBlock_8x8	ippiAdd8x8HP_16s8u_C1RS	
ippiReconBlock_H263	ippiAdd8x8HP_16s8u_C1RS	
ippiReconBlock_H263_I	ippiAdd8x8_16s8u_C1IRS	
ippiReconBlockHalfpel_MPEG4_8u	ippiAdd8x8HP_16s8u_C1R	
ippiReconMB_H263	ippiMC16x16_8u_C1	
ippiReconMB_H263_I	ippiMC16x16_8u_C1	
ippiSumNorm_VOP_MPEG4_8u16u		useless
ippiTransRecBlockCoef_inter_MPEG4	DCT and Quant functions	set of functions
ippiTransRecBlockCoef_intra_MPEG4	DCT and Quant functions	set of functions
ippiUpdateQP_MPEG4		functionality on codec level
ippiUpdateQuant_MQ_H263_1u32s_I		functionality on codec level
ippiUpdateRCModel_MPEG4		functionality on codec level
ippiVCHuffmanDecodeOne_1u32s	ippiDecodeHuffmanOne_1u32s	
ippiVCHuffmanFree_32s	ippiHuffmanTableFree_32s	
ippiVCHuffmanInitAlloc_32s	ippiHuffmanTableInitAlloc_32s	
ippiVCHuffmanInitAllocRL_32s	ippiHuffmanRunLevelTableInitAlloc_32s	
ippiZigzagInv_Horizontal_16s	ippiScanInv_16s_C1I	additional parameters
ippiZigzagInv_Vertical_16s	ippiScanInv_16s_C1I	additional parameters
ippiZigzagInvClassical_Compact_16s	ippiScanInv_16s_C1I	additional parameters
ippiZigzagInvHorizontal_Compact_16s	ippiScanInv_16s_C1I	additional parameters
ippiZigzagInvVertical_Compact_16s	ippiScanInv_16s_C1I	additional parameters

# Bibliography

---

This bibliography provides a list of publications that might be helpful to you in using the image processing subset of Intel IPP. This list is not complete; it serves only as a starting point. The books [Rogers85], [Rogers90], and [Foley90] are good resources of information on image processing and computer graphics, with mathematical formulas and code examples.

- [Akansu96] A.Akansu, M.Smith (editors). *Subband and Wavelet transform. Design and Applications*, Kluwer Academic Publishers, 1996.
- [AP922] *A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Streaming SIMD Extensions and MMX™ Instructions*, Application Note AP922, Intel Corp. Order number 742474, 1999.
- [APMF] *Fast Algorithms for Median Filtering*, Application Note, Intel Corp. Document number 79835, 2001.
- [Borge86] G. Borgefors. *Distance Transformations in Digital Images*. Computer Vision, Graphics, and Image Processing 34, 1986.
- [Bragg] Dennis Bragg. *A simple color reduction filter*, Graphic Gems III: 20–22.
- [Bouguet99] J-Y.Bouguet. *Pyramidal Implementation of the Lucas-Kanade Feature Tracker*. OpenCV Documentation, Microprocessor Research Lab, Intel Corporation, 1999.
- [Canny86] J. Canny. *A Computational Approach to Edge Detection*, IEEE Trans. on Pattern Analysis and Machine Intelligence 8(6), 1986.
- [Davis97] J.Davis and Bobick. *The Representation and Recognition of Action Using Temporal Templates*. MIT Media Lab Technical Report 402, 1997.
- [Davis99] J.Davis and G.Bradski. *Real-Time Motion Template Gradients Using Intel® Computer Vision Library*. IEEE ICCV'99 FRAME-RATE WORKSHOP, 1999.

- [Feig92] E. Feig and S. Winograd. *Fast Algorithms for the Discrete Cosine Transform*, IEEE Trans. Signal Processing, vol. 40, no. 9, pp. 2174-2193, Sep. 1992.
- [Foley90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics — Principles and Practice*, Second Edition. Addison Wesley, 1990.
- [IEC61834] IEC 61834. International Electrochemical Commission. *Specifications of Consumer-Use Digital VCRs using 6.3 mm magnetic tape (the "Blue Book" DV specification)*.
- [IEEE] *IEEE Standard Specifications for the Implementations of 8X8 Inverse Discrete Cosine Transform*, IEEE # 1180 (1997).
- [IPL] *Intel Image Processing Library Reference Manual*. Intel Corp. Order number 663791, 1999.
- [ISO11172] ISOC D 11172. Information Technology. *Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s* (1993).
- [ISO13818] ISO/IEC 13818. Information Technology. *Coding of Moving Pictures and Associated*, (11/94).
- [ISO14496] International Standard ISO/IEC 14496-2. Information Technology. *Coding of Audio-Visual Objects - Part 2: Visual*.
- [ISO14496A] ISO/IEC 14496-2:1999/Amd.1:2000(E) . Information Technology. *Coding of Audio-Visual Objects. Part2: Visual. Amendment 1: Visual Extensions*, (01/00).
- [ISO10918] International Standard ISO/IEC 10918-1, *Digital Compression and Coding of Continuous Tone Still Images*, Appendix A – *Requirements and guidelines*.
- [ISO15444] International Standard ISO/IEC 15444-1, *JPEG 2000 Image coding system*, part 1: *Core coding system*.
- [ITUH261] ITU-T Recommendation H.261. *Line transmission of non-telephone signals. Video codec for audiovisual services at p x 64 kbits* (03/93).

- 
- [ITUH263] ITU-T Recommendation H.263. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS. *Infrastructure of audiovisual services - Coding of moving video. Video coding for low bit rate communication* (02/98).
- [ITUH264] ITU-T Recommendation H.264. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS. *Infrastructure of audiovisual services - Coding of moving video. Advanced video coding for generic audiovisual services* (ITU-T Rec. H.264 | ISO/IEC 14496-10:2005)(03/05).
- [ITU709] ITU-R Recommendation BT.709, *Basic Parameter Values for the HDTV Standard for the Studio and International Programme Exchange* [formerly CCIR Rec.709] ITU, Geneva, Switzerland, 1990.
- [Jaehne95] Jaehne, Bernd. *Digital Image Processing*, 3rd Edition, Springer-Verlag, Berlin 1995.
- [Jaehne97] Jaehne, Bernd. *Practical Handbook on Image Processing for Scientific Applications*, CRC Press, New York, 1997.
- [Jack01] Jack, Keith. *Video Demystified: a Handbook for the Digital Engineer*, LLH Technology Publishing, 3rd Edition, 2001.
- [JVTG050] JVT-G050. *ITU-T Recommendation and Final Draft International Standard of Joint Video Specification* (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC) (03/03).
- [Leinhart02] R.Leinhart, J.Maydt. *An Extended Set of Haar-like Features for Rapid Object Detection*. IEEE ICIP, vol.1, pp. 900-903, Sep. 2002.
- [Lim90] Jae S.Lim. *Two-Dimensional Signal and Image Processing*, Prentice Hall, Englewood Cliffs, NJ, 1990.
- [Malvar03] H.S.Malvar, G.J.Sullivan. *Transform, Scaling & Color Space Impact of Professional Extensions*, ISO/IEC JTC/SC29/WG11 and ITU-T SG16 Q.6 Document JVT-H031, Geneva, May 2003.

- [Malvar03-1] H.S.Malvar, G.J.Sullivan. *YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range*, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, Document No.JVT-1014r3, July 2003.
- [Myler93] H.Myler, A.Weeks. *Computer Imaging Recipes in C*, Prentice Hall, 1993.
- [Puetter05] R.C.Puetter, T.R.Gosnell, and Amos Yahil. *Digital Image Recosntuction: Deblurring and Denoising*, Annual Review of Astronomy and Astrophysics, 2005.
- [Randy97] Randy Crane. *A Simplified Approach to Image Processing*, Prentice Hall PTR, 1997.
- [Rao90] K.R. Rao and P. Yip. *Discrete Cosine Transform. Algorithms, Advantages, Applications*. Academic Press, Inc, London, 1990.
- [Richardson72] W.Richardson. *Bayesian-Based Iterative Method of Image Reconstruction*. Journal of the Optical Society of America, vol.62, No.1, January 1972.
- [Ritter96] G.Ritter, J.Wilson. *Computer Vision. Algorithms in Image Algebra*. CRC Press, 1996.
- [Rogers85] David Rogers. *Procedural Elements for Computer Graphics*. McGraw-Hill, 1985.
- [Rogers90] David Rogers and J.Alan Adams. *Mathematical Elements for Computer Graphics*. McGraw-Hill, 1990.
- [Shanley98] Tom Shanley. *Pentium Pro and Pentium II System Architecture*. Addison-Wesley, 1998.
- [Schumacher] Dale A. Schumacher. *A comparison of digital halftoning techniques*, Graphic Gems III: 57–71.
- [Serra82] J.Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.
- [Thomas] Spencer W. Thomas and Rod G. Bogart. *Color dithering*, Graphic Gems II: 72–77.
- [Ulichney93] R.Ulichney. *Digital halftoning*. MIT press, 1993.

- 
- [Vincent93] L.Vincent. *Morphological Gray Scale Reconstruction in Image Analysis: Applications and Efficient Algorithms*. IEEE Transactions on Image Processing, vol.2, No.2, April 1993.
- [Viola01] P.Viola, M.J.Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features*. IEEE CVPR, 2001.
- [Wang02] Z.Wang, A.C.Bovik. *A Universal Image Quality Index*. IEEE Signal Processing Letters, vol.XX, NO.Y, March 2002.
- [Wolberg96] G.Wolberg. *Digital Image Warping*. IEEE Computer Society Press, 1996.

# Glossary

---

absolute colors	Colors specified by each pixel's coordinates in a color space. Intel Integrated Performance Primitives for image processing use images with absolute colors.
alpha channel	A color channel, also known as the opacity channel, that can be used in color models; for example, the RGBA model.
arithmetic operation	An operation that adds, subtracts, multiplies, divides, or squares the image pixel values.
color-twist matrix	A matrix used to multiply the pixel components in one color space for determining the components in another color space.
DCT	Acronym for the discrete cosine transform. See " <a href="#">Discrete Cosine Transform</a> " in Chapter 10.
dilation	A morphological operation that sets each output pixel to the minimum of the corresponding input pixel and its 8 neighbors.
dyadic operation	An operation that has two input images. It can have other input parameters as well.
element-wise operation	An element-wise operation performs the same operation on each element of a vector, or uses the elements of the same position in multiple vectors as inputs to the operation.
erosion	A morphological operation that sets each output pixel to the maximum of the corresponding input pixel and its 8 neighbors.
four-channel model	A color model that uses four color channels; for example, the RGBA color model.



gray scale image	An image characterized by a single intensity channel so that each intensity value corresponds to a certain shade of gray.
in-place operation	An operation whose output image is one of the input images.
linear filtering	In this manual, 2D convolution operations.
linear image transforms	In this manual, the discrete cosine transform (DCT).
MMX™ technology	An enhancement to the Intel architecture aimed at better performance in multimedia and communications applications. The technology uses four additional data types, eight 64-bit MMX registers, and 57 additional instructions implementing the SIMD (single instruction, multiple data) technique.
monadic operation	An operation that has a single input image. It can have other input parameters as well.
morphological operation	An erosion, dilation, or their combinations.
not-in-place operation	An operation whose output is an image other than the input image(s). <i>See</i> in-place operation.
pixel depth	The number of bits determining each channel intensity for a single pixel in the image.
pixel-oriented ordering	Storing the image information in such an order that the values of all color channels for each pixel are clustered; for example, RGBRGB... .
planar-oriented ordering	Storing the image information so that all data of one color channel follow all data of another channel, thus forming a separate “plane” for each channel; for example, RRRRRGGGGBBBBB... .
region of interest	A rectangular image region on which an operation acts (or processing occurs).
RGB	Red-green-blue. A three-channel color model that uses red, green, and blue color channels.

---

RGBA	Red-green-blue-alpha. A four-channel color model that uses red, green, blue, and alpha (or opacity) channels.
ROI	<i>See</i> region of interest.
row-major order	The default storage method for arrays in C. Memory representation is such that the rows of an array are stored contiguously. For example, for the array <code>a[3][4]</code> , the element <code>a[1][0]</code> immediately follows <code>a[0][3]</code> .
saturation	Using saturation arithmetic, when a number exceeds the data-range limit for its data type, it saturates to the upper data-range limit. For example, a signed word greater than <code>7FFFh</code> saturates to <code>7FFFh</code> . When a number is less than the lower data-range limit, it saturates to the lower data-range. For example, a signed word less than <code>8000h</code> saturates to <code>8000h</code> .
Streaming SIMD Extensions	The enhancement to Intel architecture instruction set for the next generation processors. It incorporates a group of general-purpose floating-point instructions operating on packed data, additional packed integer instructions, together with cacheability control and state management instructions. These instructions significantly improve performance of applications using compute-intensive processing of floating-point and integer data.
three-channel model	A color model that uses three color channels; for example, the RGB color model.

# *Index*

---

## **A**

- about this manual, 1-5
- about this software, 1-1
- Abs, 5-40
- AbsDiff, 5-42
- AbsDiffC, 5-43
- absolute color images, 2-18
- Add, 5-4
- Add128\_JPEG, 15-53
- Add8x8, 16-54
- Add8x8HP, 16-55
- AddBackPredPB\_H263, 16-296
- AddC, 5-7
- AddC8x8, 16-56
- AddProduct, 5-13
- AddRandGauss\_Direct, 4-34
- AddRandUniform\_Direct, 4-31
- AddRotateShift, 12-31
- AddSquare, 5-11
- AddWeighted, 5-14
- alpha channel, 2-19
- alpha composition functions
  - AlphaComp, 5-76
  - AlphaCompC, 5-78
  - AlphaPremul, 5-81
  - AlphaPremulC, 5-83
- anchor cell, in a neighborhood mask, 2-20
- And, 5-55

- AndC, 5-57
- ApplyHaarClassifier, 14-67
- arithmetic functions
  - absolute value, 5-40
  - addition, 5-4
  - addition with accumulation, 5-11
  - complement, 5-54
  - division, 5-34
  - exponential, 5-52
  - logarithm, 5-49
  - multiplication, 5-16
  - multiplication with scaling, 5-23
  - square, 5-44
  - square root, 5-46
  - subtraction, 5-27
- arithmetic operations, 5-4
- audience for this manual, 1-7
- Average16x16, 16-57
- Average8x8, 16-57

## **B**

- BGR555ToYCbCr\_JPEG, 15-12
- BGR555ToYCbCr411, 6-90
- BGR555ToYCbCr411LS\_MCU, 15-27
- BGR555ToYCbCr420, 6-87
- BGR555ToYCbCr422, 6-63
- BGR555ToYCbCr422LS\_MCU, 15-25
- BGR555ToYCbCr444LS\_MCU, 15-23
- BGR565ToYCbCr\_JPEG, 15-12

BGR565ToYCbCr411, 6-90  
 BGR565ToYCbCr411LS\_MCU, 15-27  
 BGR565ToYCbCr420, 6-87  
 BGR565ToYCbCr422, 6-63  
 BGR565ToYCbCr422LS\_MCU, 15-25  
 BGR565ToYCbCr444LS\_MCU, 15-23  
 BGRTToCbYCr422, 6-67  
 BGRTToHLS, 6-109  
 BGRTToLab, 6-99  
 BGRTToY\_JPEG, 15-5  
 BGRTToYCbCr\_JPEG, 15-10  
 BGRTToYCbCr411, 6-88  
 BGRTToYCbCr411LS\_MCU, 15-26  
 BGRTToYCbCr422, 6-61  
 BGRTToYCbCr422LS\_MCU, 15-24  
 BGRTToYCbCr444LS\_MCU, 15-22  
 BGRTToYCoCg, 6-114  
 BGRTToYCoCg\_Rev, 6-118  
 BGRTToYCrCb420, 6-86  
 BiDirWeightBlock\_H264, 16-371  
 BiDirWeightBlockImplicit\_H264, 16-372  
 bit depth, of an image, 2-18  
 bitstream parsing, 16-267, 16-283  
 BlockMatch\_Integer\_16x16\_MVFAST, 16-246  
 BlockMatch\_Integer\_16x16\_SEA, 16-243  
 borders, in neighborhood operations, 9-5  
 buffer step, 2-22  
 buffers for  
     DC/AC coefficients, 16-182  
     motion vector, 16-182  
     quantization parameters, 16-182  
     transparent status, 16-182  
     video plane, 16-181

## C

CalcGlobalMV\_MPEG4, 16-191  
 camera calibration and 3D reconstruction, 14-72  
 Canny, 14-8  
 CannyGetSize, 14-7

CbYCr422ToBGR, 6-68  
 CbYCr422ToRGB, 6-66  
 CbYCr422ToYCbCr411, 6-155  
 CbYCr422ToYCbCr420, 6-153  
 CbYCr422ToYCbCr420\_Rotate, 16-104  
 CbYCr422ToYCbCr422, 6-152  
 CbYCr422ToYCrCb420, 6-154  
 ChangeSpriteBrightness\_MPEG4, 16-191  
 chromaticity diagram, 6-7  
 CIE Lab color model, 6-21  
 CIE LUV color model, 6-21  
 CIE XYZ color model, 6-20  
 closing, 8-2  
 CMYK color model, 6-11  
 CMYKToYCKK\_JPEG, 15-15  
 CMYKToYCKK411LS\_MCU, 15-30  
 CMYKToYCKK422LS\_MCU, 15-29  
 CMYKToYCKK444LS\_MCU, 15-28  
 code samples, 1-4  
 codecs, 1-4  
 coefficient buffers, for Intra DC/AC prediction, 16-182  
 color conversion functions  
     between RGB and HLS, 6-105  
     between RGB and HSV, 6-111  
     between RGB and Lab, 6-99  
     between RGB and LUV, 6-95  
     between RGB and XYZ, 6-93  
     between RGB and YCbCr, 6-48  
     between RGB and YCbCr422, 6-58  
     between RGB and YCC, 6-102  
     between RGB and YCoCg, 6-114  
     between RGB and YCoCg-R, 6-118  
     between RGB and YUV, 6-31  
     between RGB and YUV420, 6-38  
     between RGB and YUV422, 6-34  
     bit reduction, 6-138  
     color to gray scale, 6-123  
     color twist, 6-178, 6-179  
     gamma correction, 6-183  
 color conversion functions, for JPEG codec, 15-3  
 color gamut, 6-7  
 color median filter, 9-29

---

- color spaces
  - CIE Lab, 6-21
  - CIE LUV, 6-21
  - CIE XYZ, 6-20
  - CMYK, 6-11
  - HLS, 6-17
  - HSV, 6-17
  - Photo YCC, 6-14
  - RGB, 6-9
  - YCbCr, 6-13
  - YCKK, 6-13
  - YCoCg, 6-16
  - YCoCg-R, 6-16
  - YUV, 6-11
- ColorToGray, 6-124
- ColorTwist, 6-179
- ColorTwist32f, 6-181
- combined quantization, DCT, and level shift functions, 15-48
- Compare, 7-19
- compare functions, 7-19
- CompareC, 7-20
- CompareEqualEps, 7-23
- CompareEqualEpsC, 7-24
- Complement, 5-54
- compressed macroblock, 16-147
- compressed segment, 16-147
- ComputeTextureErrorBlock, 16-250
- ComputeTextureErrorBlock\_SAD, 16-249
- concepts
  - image data types and ranges, 2-18
  - major operation models and key parameters, 2-20
  - of IPP, 2-1
  - structures and enumerators, 2-13
- conventions
  - font, 1-7
  - function naming, 2-2
  - notational, 1-7
- Convert, 4-2
- ConvFull, 9-61
- ConvValid, 9-64
- Copy, 4-11
- Copy16x16, 16-51
- Copy16x16HP, 16-52
- Copy16x16QP\_MPEG4, 16-184
- Copy16x8HP, 16-52
- Copy16x8QP\_MPEG4, 16-184
- Copy8x4HP, 16-52
- Copy8x8, 16-51
- Copy8x8HP, 16-52
- Copy8x8QP\_MPEG4, 16-184
- CopyConstBorder, 4-15
- CopyReplicateBorder, 4-18
- CopySubpix, 4-23
- CopySubpixIntersect, 4-24
- CopyWrapBorder, 4-20
- correction of camera lens distortion
  - CreateMapCameraUndistort, 14-75
  - UndistortGetSize, 14-72
  - UndistortRadial, 14-73
- CountInRange, 11-30
- CountZeros8x8, 16-161
- CreateMapCameraUndistort, 14-75
- Cross-Architecture Alignment, 1-2
- CrossCorrFull\_Norm, 11-90
- CrossCorrFull\_NormLevel, 11-96
- CrossCorrSame\_Norm, 11-92
- CrossCorrValid\_Norm, 11-94
- CrossCorrValid\_NormLevel, 11-100
- cross-platform applications, 2-2

**D**

- data exchange functions
  - adding Gaussian noise, 4-34
  - adding uniform noise, 4-31
  - conversion, 4-2
  - CopyConstBorder, 4-15
  - copying, 4-11
  - CopyReplicateBorder, 4-18
  - CopyWrapBorder, 4-20
  - duplicate gray scale image, 4-27
  - initializing, 4-9

---

scaling, 4-6  
swap channels, 4-30  
transpose image, 4-28  
DCT block, 16-144  
DCT2x4x8Fwd, 16-160  
DCT2x4x8Inv, 16-157  
DCT8x8Fwd, 10-50  
DCT8x8FwdLS, 10-54  
DCT8x8Inv, 10-52  
DCT8x8Inv\_AANTransposed\_16s\_C1R, 16-129  
DCT8x8Inv\_AANTransposed\_16s\_P2C2R, 16-131  
DCT8x8Inv\_AANTransposed\_16s8u\_C1R, 16-130  
DCT8x8Inv\_AANTransposed\_16s8u\_P2C2R, 16-132  
DCT8x8InvLSClip, 10-55  
DCTFwd, 10-47  
DCTFwdFree, 10-44  
DCTFwdGetBufSize, 10-45  
DCTFwdInitAlloc, 10-42  
DCTInv, 10-49  
DCTInvFree, 10-45  
DCTInvGetBufSize, 10-46  
DCTInvInitAlloc, 10-43  
DCTQuantFwd8x8\_JPEG, 15-48  
DCTQuantFwd8x8LS\_JPEG, 15-49  
DCTQuantInv8x8\_JPEG, 15-50  
DCTQuantInv8x8LS\_JPEG, 15-51  
DecodeBlockCoef\_Inter\_H263, 16-287  
DecodeBlockCoef\_Inter\_MPEG4, 16-226  
DecodeBlockCoef\_Intra\_H263, 16-286  
DecodeBlockCoef\_Intra\_MPEG4, 16-224  
DecodeCAEInterH\_MPEG4, 16-231  
DecodeCAEInterV\_MPEG4, 16-231  
DecodeCAEIntraH\_MPEG4, 16-230  
DecodeCAEIntraV\_MPEG4, 16-230  
DecodeCAVLCChromaDcCoeffs\_H264, 16-337  
DecodeCAVLCCoeffs\_H264, 16-335  
DecodeCBProgrAttach\_JPEG2K, 15-159  
DecodeCBProgrFree\_JPEG2K, 15-158  
DecodeCBProgrGetStateSize, 15-156

DecodeCBProgrInit\_JPEG2K, 15-157  
DecodeCBProgrInitAlloc\_JPEG2K, 15-157  
DecodeCodeBlock\_JPEG2K, 15-154  
DecodeCoeffsInter\_H261, 16-271  
DecodeCoeffsInter\_MPEG4, 16-217  
DecodeCoeffsInterRVLCBack\_MPEG4, 16-219  
DecodeCoeffsIntra\_H261, 16-270  
DecodeCoeffsIntra\_H263, 16-289  
DecodeCoeffsIntra\_MPEG4, 16-215  
DecodeCoeffsIntraRVLCBack\_MPEG4, 16-216  
DecodeDCIntra\_H263, 16-288  
DecodeDCIntra\_MPEG4, 16-214  
DecodeExpGolombOne\_H264, 16-338  
DecodeGetBufSize\_JPEG2K, 15-154  
DecodeHuffman8x8\_ACFirst\_JPEG, 15-104  
DecodeHuffman8x8\_ACRrefine\_JPEG, 15-105  
DecodeHuffman8x8\_DCFirst\_JPEG, 15-101  
DecodeHuffman8x8\_DCRrefine\_JPEG, 15-103  
DecodeHuffman8x8\_Direct\_JPEG, 15-100  
DecodeHuffman8x8\_JPEG, 15-98  
DecodeHuffmanOne, 16-20  
DecodeHuffmanOne\_JPEG, 15-113  
DecodeHuffmanPair, 16-21  
DecodeHuffmanSpecFree\_JPEG, 15-95  
DecodeHuffmanSpecGetBufSize\_JPEG, 15-92  
DecodeHuffmanSpecInit\_JPEG, 15-93  
DecodeHuffmanSpecInitAlloc\_JPEG, 15-94  
DecodeHuffmanStateFree\_JPEG, 15-98  
DecodeHuffmanStateGetBufSize\_JPEG, 15-96  
DecodeHuffmanStateInit\_JPEG, 15-96  
DecodeHuffmanStateInitAlloc\_JPEG, 15-97  
DecodeMV\_BVOP\_Backward\_MPEG4, 16-196  
DecodeMV\_BVOP\_Direct\_MPEG4, 16-198  
DecodeMV\_BVOP\_DirectSkip\_MPEG4, 16-200  
DecodeMV\_BVOP\_Forward\_MPEG4, 16-195  
DecodeMV\_BVOP\_Interpolate\_MPEG4, 16-197  
DecodeMVS\_MPEG4, 16-233  
DecodePadMV\_PVOP\_MPEG4, 16-193  
DecodeVLCZigzag\_Inter\_MPEG4, 16-223

- DecodeVLCZigzag\_IntraACVLC\_MPEG4, 16-221
- DecodeVLCZigzag\_IntraDCVLC\_MPEG4, 16-221
- DeconvFFT, 9-69
- DeconvFFTFree, 9-68
- DeconvFFTInitAlloc, 9-67
- DeconvLR, 9-71
- DeconvLRFree, 9-71
- DeconvLRInitAlloc, 9-70
- deconvolution functions
  - DeconvFFT, 9-69
  - DeconvFFTFree, 9-68
  - DeconvFFTInitAlloc, 9-67
  - DeconvLR, 9-71
  - DeconvLRFree, 9-71
  - DeconvLRInitAlloc, 9-70
- deinterlace filtering functions
  - YCbCr420ToYCbCr420\_Filter, 6-163
- DeinterlaceFilterTriangle, 16-109
- deinterlacing, 16-109
- dequantization, of the DCT coefficients, 15-47
- DequantTransformResidual\_H264, 16-341
- DequantTransformResidual\_SISP\_H264, 16-346
- DequantTransformResidualAndAdd\_H264, 16-343
- DFTFree, 10-17
- DFTFwd, 10-19
- DFTGetBufSize, 10-18
- DFTInitAlloc, 10-16
- DFTInv, 10-22
- DiffPredFirstRow\_JPEG, 15-107
- DiffPredRow\_JPEG, 15-108
- Dilate, 8-11
- Dilate3x3, 8-7
- DilateBorderReplicate, 8-19
- dilation, 8-2
- direction, in DC/AC prediction, 16-179
- DistanceTransform, 14-16
- Div, 5-34
- DivC, 5-37
- DownsampleFour\_H263, 16-301
- DownsampleFour16x16\_H263, 16-320
- downsampling, image, 6-24
- Dup, 4-27
- DV, 16-144
- DV decoder functions
  - inverse discrete cosine transformation
    - DCT2x4x8Inv, 16-157
  - inverse quantization
    - QuantInv\_DV, 16-156
  - variable length decoding
    - FreeHuffmanTable\_DV, 16-156
    - HuffmanDecode Segment\_DV, 16-154
    - InitAllocHuffmanTable\_DV, 16-153
- DV encoder functions
  - color conversion
    - YCrCb411ToYCbCr422\_5MBDV, 16-162
    - YCrCb411ToYCbCr422\_EdgeDV, 16-163
    - YCrCb411ToYCbCr422\_ZoomOut2\_5MBDV, 16-162
    - YCrCb411ToYCbCr422\_ZoomOut2\_EdgeDV, 16-163
    - YCrCb411ToYCbCr422\_ZoomOut4\_5MBDV, 16-162
    - YCrCb411ToYCbCr422\_ZoomOut4\_EdgeDV, 16-163
    - YCrCb411ToYCbCr422\_ZoomOut8\_5MBDV, 16-162
    - YCrCb411ToYCbCr422\_ZoomOut8\_EdgeDV, 16-163
    - YCrCb420ToYCbCr422\_5MBDV, 16-164
    - YCrCb420ToYCbCr422\_ZoomOut2\_5MBDV, 16-164
    - YCrCb420ToYCbCr422\_ZoomOut4\_5MBDV, 16-164
    - YCrCb420ToYCbCr422\_ZoomOut8\_5MBDV, 16-164
    - YCrCb422ToYCbCr422\_5MBDV, 16-166
    - YCrCb422ToYCbCr422\_ZoomOut2\_5MBDV, 16-166
    - YCrCb422ToYCbCr422\_ZoomOut4\_5MBDV, 16-166
    - YCrCb422ToYCbCr422\_ZoomOut8\_5MBDV, 16-166
- discrete cosine transformation
  - CountZeros8x8, 16-161
  - DCT2x4x8Frw, 16-160

---

## E

EigenValsVecs, 14-10  
EigenValsVecsGetBufferSize, 14-9  
EncodeChromaDcCoeffsCAVLC\_H264\_16s, 16-429  
EncodeCoeffsCAVLC\_H264\_16s, 16-425  
EncodeCoeffsInter\_H261, 16-278  
EncodeCoeffsInter\_H263, 16-316  
EncodeCoeffsInter\_MPEG4, 16-257  
EncodeCoeffsIntra\_H261, 16-277  
EncodeCoeffsIntra\_H263, 16-315  
EncodeCoeffsIntra\_MPEG4, 16-256  
EncodeDCIntra\_H263, 16-314  
EncodeDCIntra\_MPEG4, 16-255  
EncodeFree\_JPEG2K, 15-146  
EncodeGetDist\_JPEG2K, 15-153  
EncodeGetRate\_JPEG2K, 15-152  
EncodeGetTermPassLen\_JPEG2K, 15-151  
EncodeHuffman8x8\_ACFirst\_JPEG, 15-90  
EncodeHuffman8x8\_ACRefine\_JPEG, 15-91  
EncodeHuffman8x8\_DCFirst\_JPEG, 15-87  
EncodeHuffman8x8\_DCRefine\_JPEG, 15-88  
EncodeHuffman8x8\_Direct\_JPEG, 15-82  
EncodeHuffman8x8\_JPEG, 15-81  
EncodeHuffmanOne\_JPEG, 15-112  
EncodeHuffmanRawTableInit\_JPEG, 15-74  
EncodeHuffmanSpecFree\_JPEG, 15-78  
EncodeHuffmanSpecGetBufSize\_JPEG, 15-75  
EncodeHuffmanSpecInit\_JPEG, 15-76  
EncodeHuffmanSpecInitAlloc\_JPEG, 15-77  
EncodeHuffmanStateFree\_JPEG, 15-80  
EncodeHuffmanStateGetBufSize\_JPEG, 15-78  
EncodeHuffmanStateInit\_JPEG, 15-79  
EncodeHuffmanStateInitAlloc\_JPEG, 15-80  
EncodeInitAlloc\_JPEG2K, 15-145  
EncodeLoadCodeBlock\_JPEG2K, 15-146  
EncodeMV\_MPEG4, 16-264  
EncodeStoreBits\_JPEG2K, 15-149  
EncodeVLCZigzag\_Inter\_MPEG4, 16-259  
EncodeVLCZigzag\_IntraACVLC\_MPEG4, 16-258

EncodeVLCZigzag\_IntraDCVLC\_MPEG4, 16-258  
enumerators, 2-13  
Erode, 8-13  
Erode3x3, 8-9  
ErodeBorderReplicate, 8-21  
erosion, 8-2  
error handling, 2-8  
error messages, 2-9  
Exp, 5-52  
ExpandFrame\_H263, 16-297  
ExpandPlane\_H264, 16-353

## F

feature detection functions  
    Canny, 14-8  
    CannyGetSize, 14-7  
    EigenValsVecs, 14-10  
    EigenValsVecsGetBufferSize, 14-9  
    MinEigenVal, 14-14  
    MinEigenValGetBufferSize, 14-13  
FFTFree, 10-7  
FFTFwd, 10-9  
FFTGetBufSize, 10-8  
FFTInitAlloc, 10-6  
FFTIInv, 10-14  
FillterScharrHorizGetBufferSize, 9-99  
Filter, 9-31  
Filter32f, 9-34  
Filter8x8\_H261, 16-279  
FilterBlockBoundaryHorEdge\_H263, 16-305  
FilterBlockBoundaryVerEdge\_H263, 16-305  
FilterBox, 9-8  
FilterColumn, 9-36  
FilterColumn32f, 9-38  
FilterColumnPipeline, 9-52  
FilterColumnPipeline\_Low, 9-54  
FilterColumnPipelineGetBufferSize, 9-45  
FilterColumnPipelineGetBufferSize\_Low, 9-46  
FilterDeblocking16x16\_HorEdge\_H263, 16-308



FilterDeblocking16x16\_VerEdge\_H263, 16-308  
FilterDeblocking8x8HorEdge\_H263, 16-306  
FilterDeblocking8x8HorEdge\_MPEG4, 16-227  
FilterDeblocking8x8VerEdge\_H263, 16-306  
FilterDeblocking8x8VerEdge\_MPEG4, 16-227  
FilterDeblockingChroma\_HorEdge\_H264, 16-413  
FilterDeblockingChroma\_VerEdge\_H264, 16-409  
FilterDeblockingChroma\_VerEdge\_MBAFF\_H264, 16-412  
FilterDeblockingLuma\_HorEdge\_H264, 16-407  
FilterDeblockingLuma\_VerEdge\_H264, 16-404  
FilterDeblockingLuma\_VerEdge\_MBAFF\_H264, 16-406  
FilterDeringingSmooth8x8\_MPEG4, 16-229  
FilterDeringingThreshold\_MPEG4, 16-228  
FilterGauss, 9-92  
FilterHipass, 9-94  
filtering functions  
    box filter, 9-8  
    color median filter, 9-29  
    column filter, 9-36  
    Gaussian filter, 9-92  
    general rectangular filter, 9-31  
    highpass filter, 9-94  
    Laplacian filter, 9-91  
    lowpass filter, 9-95  
    max filter, 9-15  
    max filter border replication, 9-20  
    max filter working buffer size, 9-17  
    median filter, 9-23  
    min filter, 9-12  
    min filter border replication, 9-18  
    min filter working buffer size, 9-16  
    Prewitt filter, 9-74  
    Roberts filter, 9-88  
    row filter, 9-39  
    Scharr filter, 9-78  
    sharpening filter, 9-97  
    Sobel filter, 9-80, 9-81  
FilterLaplace, 9-91  
FilterLaplacianBorder, 9-122  
FilterLaplacianGetBufferSize, 9-106  
FilterLowpass, 9-95  
FilterLowpassBorder, 9-124  
FilterLowpassGetBufferSize, 9-107  
FilterMax, 9-15  
FilterMaxBorderReplicate, 9-20  
FilterMaxGetBufferSize, 9-17  
FilterMedian, 9-23  
FilterMedianColor, 9-29  
FilterMedianCross, 9-28  
FilterMedianHoriz, 9-25  
FilterMedianVert, 9-26  
FilterMin, 9-12  
FilterMinBorderReplicate, 9-18  
FilterMinGetBufferSize, 9-16  
FilterPrewittHoriz, 9-74  
FilterPrewittVert, 9-75  
FilterRobertsDown, 9-88  
FilterRobertsUp, 9-89  
FilterRow, 9-39  
FilterRow32f, 9-42  
FilterRowBorderPipeline, 9-47  
FilterRowBorderPipeline\_Low, 9-50  
FilterRowBorderPipelineGetBufferSize, 9-43  
FilterRowBorderPipelineGetBufferSize\_Low, 9-44  
FilterScharrHoriz, 9-78  
FilterScharrHorizBorder, 9-109  
FilterScharrVert, 9-79  
FilterScharrVertBorder, 9-110  
FilterScharrVertGetBufferSize, 9-100  
FilterSharpen, 9-97  
FilterSobelCross, 9-87  
FilterSobelCrossBorder, 9-120  
FilterSobelCrossGetBufferSize, 9-105  
FilterSobelHoriz, 9-80  
FilterSobelHorizBorder, 9-112  
FilterSobelHorizGetBufferSize, 9-101  
FilterSobelHorizMask, 9-81  
FilterSobelHorizSecond, 9-84  
FilterSobelHorizSecondBorder, 9-116  
FilterSobelHorizSecondGetBufferSize, 9-103

---

FilterSobelNegVertBorder, 9-114  
FilterSobelNegVertGetBufferSize, 9-102  
FilterSobelVert, 9-82  
FilterSobelVertBorder, 9-114  
FilterSobelVertGetBufferSize, 9-102  
FilterSobelVertMask, 9-82  
FilterSobelVertSecond, 9-85  
FilterSobelVertSecondBorder, 9-118  
FilterSobelVertSecondGetBufferSize, 9-104  
FilterWiener, 9-58  
FilterWienerGetBufferSize, 9-57  
FindMVPred\_MPEG4, 16-263  
fixed filters, 9-73  
fixed filters with border, 9-99  
flood fill functions  
    FloodFill, 14-25  
    FloodFill\_Grad, 14-27  
    FloodFill\_Range, 14-29  
    FloodFillGetSize, 14-23  
    FloodFillGetSize\_Grad, 14-24  
FloodFill, 14-25  
FloodFill\_Grad, 14-27  
FloodFill\_Range, 14-29  
FloodFillGetSize, 14-23  
FloodFillGetSize\_Grad, 14-24  
font conventions, 1-7  
format conversion functions  
    CbYCr422ToYCbCr411, 6-155  
    CbYCr422ToYCbCr420, 6-153  
    CbYCr422ToYCbCr422, 6-152  
    CbYCr422ToYCrCb420, 6-154  
    YCbCr411, 6-171  
    YCbCr411ToYCbCr420, 6-175  
    YCbCr411ToYCbCr422, 6-173  
    YCbCr411ToYCrCb420, 6-177  
    YCbCr411ToYCrCb422, 6-174  
    YCbCr420, 6-156  
    YCbCr420ToCbYCr422, 6-161  
    YCbCr420ToYCbCr411, 6-165  
    YCbCr420ToYCbCr422, 6-157  
    YCbCr420ToYCbCr422\_Filter, 6-159  
    YCbCr420ToYCrCb420, 6-162

    YCbCr420ToYCrCb420\_Filter, 6-163  
    YCbCr422, 6-141  
    YCbCr422ToCbYCr422, 6-143  
    YCbCr422ToYCbCr411, 6-147  
    YCbCr422ToYCbCr420, 6-144  
    YCbCr422ToYCrCb420, 6-146  
    YCbCr422ToYCrCb422, 6-142  
    YCrCb420ToCbYCr422, 6-168  
    YCrCb420ToYCbCr411, 6-170  
    YCrCb420ToYCbCr420, 6-169  
    YCrCb420ToYCbCr422, 6-166  
    YCrCb420ToYCbCr422\_Filter, 6-167  
    YCrCb422ToYCbCr411, 6-151  
    YCrCb422ToYCbCr420, 6-150  
    YCrCb422ToYCbCr422, 6-149  
FrameFieldSAD16x16, 16-94  
FreeHuffmanTable\_DV, 16-156  
function  
    context, 2-18  
    descriptions, 1-6  
    parameters, 2-6  
    prototypes, 2-6

## G

gamma correction, 6-7  
GammaFwd, 6-183  
GammaInv, 6-186  
GenScaleLevel8x8\_H264, 16-424  
GenSobelKernel, 9-108  
geometric transform functions  
    affine warp, 12-41  
    bilinear warp, 12-66  
    mirroring, 12-21  
    perspective warp, 12-54  
    remapping, 12-23  
    resizing, 12-5  
    rotation, 12-26  
    shear, 12-36  
GetAffineBound, 12-52  
GetAffineQuad, 12-51  
GetAffineTransform, 12-53  
GetBilinearBound, 12-76

GetBilinearQuad, 12-75  
GetBilinearTransform, 12-77  
GetCentralMoment, 11-50  
GetDiff16x16, 16-60  
GetDiff16x16B, 16-67  
GetDiff16x8, 16-61  
GetDiff16x8B, 16-69  
GetDiff4x4, 16-66  
GetDiff8x16, 16-64  
GetDiff8x16B, 16-71  
GetDiff8x4, 16-65  
GetDiff8x4B, 16-72  
GetDiff8x8, 16-62  
GetDiff8x8B, 16-70  
GetDistanceTransformMask, 14-19  
GetHaarClassifierSize, 14-35  
GetHuffmanStatistics8x8\_ACFirst\_JPEG, 15-85  
GetHuffmanStatistics8x8\_ACRrefine\_JPEG, 15-86  
GetHuffmanStatistics8x8\_DCFirst\_JPEG, 15-84  
GetHuffmanStatistics8x8\_JPEG, 15-83  
GetHuffmanStatisticsOne\_JPEG, 15-111  
GetHuMoments, 11-52  
GetLibVersion, 3-2  
GetNormalizedCentralMoment, 11-51  
GetNormalizedSpatialMoment, 11-48  
GetPerspectiveBound, 12-63  
GetPerspectiveQuad, 12-62  
GetPerspectiveTransform, 12-64  
GetPyramidDownROI, 14-52  
GetResizeFract, 12-16  
GetRotateBound, 12-33  
GetRotateQuad, 12-32  
GetRotateShift, 12-30  
GetShearBound, 12-40  
GetShearQuad, 12-39  
GetSpatialMoment, 11-47  
GradientColorToGray, 14-21  
GrayDilateBorder, 8-39, 8-40

## H

H.263 decoder frame expansion functions  
ExpandFrame\_H263, 16-297  
H.263 decoder inverse quantization  
QuantInvInter\_H263, 16-294  
QuantInvIntra\_H263, 16-292  
H.263 decoder prediction functions  
AddBackPredPB\_H263, 16-296  
H.263 decoder resampling functions  
DownsampleFour\_H263, 16-301  
Resample\_H263, 16-298  
SpatialInterpolation\_H263, 16-303  
UnsampleFour\_H263, 16-300  
UnsampleFour8x8\_H263, 16-302  
H.263 Encoder functions  
DownsampleFour16x16\_H263, 16-320  
EncodeCoeffsInter\_H263, 16-316  
EncodeCoeffsIntra\_H263, 16-315  
EncodeDCIntra\_H263, 16-314  
QuantInter\_H263, 16-319  
QuantIntra\_H263, 16-318  
H.263+ functions  
DownsampleFour\_H263, 16-301  
ExpandFrame\_H263, 16-297  
FilterBlockBoundaryHorEdge\_H263, 16-305  
FilterBlockBoundaryVerEdge\_H263, 16-305  
FilterDeblocking16x16HorEdge\_H263, 16-308  
FilterDeblocking16x16VerEdge\_H263, 16-308  
FilterDeblocking8x8HorEdge\_H263, 16-306  
FilterDeblocking8x8VerEdge\_H263, 16-306  
ReconstructCoeffsInter\_H263, 16-311  
ReconstructCoeffsIntra\_H263, 16-309  
Resample\_H263, 16-298  
SpatialInterpolation\_H263, 16-303  
UnsampleFour\_H263, 16-300  
UnsampleFour8x8\_H263, 16-302  
H.263+ middle-level functions  
ReconstructCoeffsInter\_H263, 16-311  
ReconstructCoeffsIntra\_H263, 16-309  
H.263+ VLC decoding  
DecodeBlockCoef\_Inter\_H263, 16-287  
DecodeBlockCoef\_Intra\_H263, 16-286  
DecodeCoeffsInter\_H263, 16-291  
DecodeCoeffsIntra\_H263, 16-289

---

DecodeDCIntra\_H263, 16-288

H263 decoder boundary filtering functions

- FilterBlockBounderyHorEdge\_H263, 16-305
- FilterBlockBounderyVerrEdge\_H263, 16-305
- FilterDeblocking16x16HorEdge\_H263, 16-308
- FilterDeblocking16x16VerEdge\_H263, 16-308
- FilterDeblocking8x8HorEdge\_H263, 16-306
- FilterDeblocking8x8VerEdge\_H263, 16-306

H264 Decoder functions

- BiDirWeightBlock\_H264, 16-371
- BiDirWeightBlockImplicit\_H264, 16-372
- DequantTransformResidual\_H264, 16-341
- DequantTransformResidual\_SISP\_H264, 16-346
- DequantTransformResidualAndAdd\_H264, 16-343
- ExpandPlane\_H264, 16-353
- FilterDeblockingChroma\_HorEdge\_H264, 16-413
- FilterDeblockingChroma\_VerEdge\_H264, 16-409
- FilterDeblockingChroma\_VerEdge\_MBAFF\_H264, 16-412
- FilterDeblockingLuma\_HorEdge\_H264, 16-407
- FilterDeblockingLuma\_VerEdge\_H264, 16-404
- FilterDeblockingLuma\_VerEdge\_MBAFF\_H264, 16-406

high profile functions, 16-380, 16-382, 16-384, 16-391, 16-392, 16-394, 16-395, 16-397, 16-398, 16-402

- InterpolateBlock\_H264, 16-367
- InterpolateChroma\_H264, 16-361
- InterpolateChromaBottom\_H264, 16-365
- InterpolateChromaTop\_H264, 16-363
- InterpolateLuma\_H264, 16-354
- InterpolateLumaBottom\_H264, 16-358
- InterpolateLumaTop\_H264, 16-356
- PredictIntra\_16x16\_H264, 16-350
- PredictIntra\_4x4\_H264, 16-348
- PredictIntraChroma8x8\_H264, 16-351
- ReconstructChromaInter4x4MB\_H264, 16-380
- ReconstructChromaInterMB\_H264, 16-374
- ReconstructChromaIntra4x4MB\_H264, 16-384
- ReconstructChromaIntraHalves4x4MB\_H264, 16-382
- ReconstructChromaIntraHalvesMB\_H264, 16-376
- ReconstructChromaIntraMB\_H264, 16-378
- ReconstructLumaInter4x4MB\_H264, 16-391
- ReconstructLumaInter8x8MB\_H264, 16-395
- ReconstructLumaInterMB\_H264, 16-386

- ReconstructLumaIntra16x16MB\_H264, 16-400, 16-402
- ReconstructLumaIntra4x4MB\_H264, 16-394
- ReconstructLumaIntra8x8MB\_H264, 16-398
- ReconstructLumaIntraHalf4x4MB\_H264, 16-392
- ReconstructLumaIntraHalf8x8MB\_H264, 16-397
- ReconstructLumaIntraHalfMB\_H264, 16-387
- ReconstructLumaIntraMB\_H264, 16-389
- TransformDequantChromaDC\_H264, 16-340
- TransformDequantChromaDC\_SISP\_H264, 16-347
- TransformDequantLumaDC\_H264, 16-339
- TransformPrediction\_H264, 16-345
- UniDirWeightBlock\_H264, 16-369
- WeightedAverage\_H264, 16-368

H264 Encoder functions

- EncodeChromaDcCoeffsCAVLC\_H264\_16s, 16-429
- EncodeCoeffsCAVLC\_H264\_16s, 16-425
- GenScaleLevel8x8\_H264, 16-424
- QuantLuma8x8\_H264, 16-423
- QuantLuma8x8Inv\_H264, 16-430
- TransformLuma8x8Fwd\_H264, 16-422
- TransformLuma8x8InvAddPred\_H264, 16-431
- TransformQuantChromaDC\_H264, 16-417
- TransformQuantLumaDC\_H264, 16-419
- TransformQuantResidual\_H264, 16-421

Haar Features, 14-60

HaarClassifierFree, 14-65

HaarClassifierInitAlloc, 14-62

hardware and software requirements, 1-2

Hint arguments for Image Moment functions, 11-42

HistogramEven, 11-27

HistogramRange, 11-24

HLS color model, 6-17

HLSToBGR, 6-110

HLSToRGB, 6-107

horizontal sampling, in a JPEG codec, 15-54

HSV color model, 6-17

HSVToRGB, 6-113

Huffman codec functions, 15-72

HuffmanDecodeSegment\_DV, 16-154

HuffmanRunLevelTableInitAlloc, 16-19

HuffmanTableFree, 16-22

HuffmanTableInitAlloc, 16-18

**I**

- ICTFwd\_JPEG2K, 15-167
- ICTInv\_JPEG2K, 15-169
- image area extension, 9-5
- image data
  - descriptors, 2-5
  - range, 2-19
  - storage, 2-18
- image format conversion, 6-140
- ImageJaehne, 4-35
- ImageRamp, 4-37
- InitAllocHuffmanTable\_DV, 16-153
- initialization functions
  - creating a test image, 4-35
- Integral, 11-7
- Intel Performance Library Suite, 2-1
- Intel Performance Primitives software, 1-1
- intensity transform functions
  - LUT, 6-125
  - LUT\_Cubic, 6-132
  - LUT\_Linear, 6-129
  - LUTPalette, 6-137
- INTER macroblock decoding, 16-269, 16-285
- interfield compression, 16-144
- InterpolateAverage16x16, 16-53
- InterpolateAverage16x8, 16-53
- InterpolateAverage8x4, 16-53
- InterpolateBlock\_H264, 16-367
- InterpolateChroma\_H264, 16-361
- InterpolateChromaBottom\_H264, 16-365
- InterpolateChromaTop\_H264, 16-363
- InterpolateLuma\_H264, 16-354
- InterpolateLumaBottom\_H264, 16-358
- InterpolateLumaTop\_H264, 16-356
- interpolation method, 12-3
- INTRA macroblock decoding, 16-268, 16-284
- inverse discrete cosine transformation, 16-129
- ipiFloodFill, 14-25
- IPP functionality
  - arithmetic and logical operations, 5-1
  - color conversion, 6-1
  - computer vision, 14-1
  - data exchange functions, 4-1
  - filtering operations, 9-1
  - geometric transforms, 12-1
  - H.261 video decoder, 16-267
  - H.261 video encoder, 16-276
  - H.263 video decoder, 16-283
  - image statistics, 11-1
  - intensity transformation, 6-125
  - linear transforms, 10-1
  - morphological operations, 8-1
  - MPEG-4 video decoder, 16-172
  - MPEG-4 video encoder, 16-238
  - threshold and compare functions, 7-1
  - wavelet transforms, 13-1
- ippGetStatusString, 3-4
- ippiAbs, 5-40
- ippiAbsDiff, 5-42
- ippiAbsDiffC, 5-43
- ippiAdd, 5-4
- ippiAdd128\_JPEG, 15-53
- ippiAdd8x8, 16-54
- ippiAdd8x8HP, 16-55
- ippiAddBackPredPB\_H263, 16-296
- ippiAddC, 5-7
- ippiAddC8x8, 16-56
- ippiAddProduct, 5-13
- ippiAddRandGauss\_Direct, 4-34
- ippiAddRandUniform\_Direct, 4-31
- ippiAddRotateShift, 12-31
- ippiAddSquare, 5-11
- ippiAddWeighted, 5-14
- ippiAlphaComp, 5-76
- ippiAlphaCompC, 5-78
- ippiAlphaPremul, 5-81
- ippiAlphaPremulC, 5-83
- ippiAnd, 5-55
- ippiAndC, 5-57
- ippiAverage16x16, 16-57
- ippiAverage8x8, 16-57

---

ippiBGR555ToYCbCr\_JPEG, 15-12  
ippiBGR555ToYCbCr411, 6-90  
ippiBGR555ToYCbCr411LS\_MCU, 15-27  
ippiBGR555ToYCbCr420, 6-87  
ippiBGR555ToYCbCr422, 6-63  
ippiBGR555ToYCbCr422LS\_MCU, 15-25  
ippiBGR555ToYCbCr444LS\_MCU, 15-23  
ippiBGR565ToYCbCr411, 6-90  
ippiBGR565ToYCbCr411LS\_MCU, 15-27  
ippiBGR565ToYCbCr420, 6-87  
ippiBGR565ToYCbCr422, 6-63  
ippiBGR565ToYCbCr422LS\_MCU, 15-25  
ippiBGR565ToYCbCr444LS\_MCU, 15-23  
ippiBGRTToCbYCr422, 6-67  
ippiBGRTToHLS, 6-109  
ippiBGRTToLab, 6-99  
ippiBGRTToY\_JPEG, 15-5  
ippiBGRTToYCbCr\_JPEG, 15-10  
ippiBGRTToYCbCr411LS\_MCU, 15-26  
ippiBGRTToYCbCr422, 6-61  
ippiBGRTToYCbCr422LS\_MCU, 15-24  
ippiBGRTToYCbCr444LS\_MCU, 15-22  
ippiBGRTToYCoCg, 6-114  
ippiBGRTToYCoCg\_Rev, 6-118  
ippiBGRTToYCrCb420, 6-86  
ippiBiDirWeightBlock\_H264, 16-371  
ippiBiDirWeightBlockImplicit\_H264, 16-372  
ippiBlockMatch\_Integer\_16x16\_MVFAST, 16-246  
ippiBlockMatch\_Integer\_16x16\_SEA, 16-243  
ippiCalcGlobalMV\_MPEG4, 16-191  
ippiCanny, 14-8  
ippiCannyGetSize, 14-7  
ippiCbYCr422ToBGR, 6-68  
ippiCbYCr422ToRGB, 6-66  
ippiCbYCr422ToYCbCr411, 6-155  
ippiCbYCr422ToYCbCr420, 6-153  
ippiCbYCr422ToYCbCr422, 6-152  
ippiCbYCr422ToYCbCr422\_Rotate, 16-104  
ippiCbYCr422ToYCrCb420, 6-154  
ippiChangeSpriteBrightness\_MPEG4, 16-191  
ippiCMYKToYCCCK\_JPEG, 15-15  
ippiCMYKToYCCCK411LS\_MCU, 15-30  
ippiCMYKToYCCCK422LS\_MCU, 15-29  
ippiCMYKToYCCCK444LS\_MCU, 15-28  
ippiColorToGray, 6-124  
ippiColorTwist, 6-179  
ippiColorTwist32f, 6-181  
ippiCompare, 7-19  
ippiCompareC, 7-20  
ippiCompareEqualEps, 7-23  
ippiCompareEqualEpsC, 7-24  
ippiComplement, 5-54  
ippiComputeTextureErrorBlock, 16-250  
ippiComputeTextureErrorBlock\_SAD, 16-249  
ippiConvert, 4-2  
ippiConvFull, 9-61  
ippiConvValid, 9-64  
ippiCopy, 4-11  
ippiCopy16x16, 16-51  
ippiCopy16x16HP, 16-52  
ippiCopy16x16QP\_MPEG4, 16-184  
ippiCopy16x8HP, 16-52  
ippiCopy16x8QP\_MPEG4, 16-184  
ippiCopy8x4HP, 16-52  
ippiCopy8x8, 16-51  
ippiCopy8x8HP, 16-52  
ippiCopy8x8QP\_MPEG4, 16-184  
ippiCopyConstBorder, 4-15  
ippiCopyReplicateBorder, 4-18  
ippiCopySubpix, 4-23  
ippiCopySubpixIntersect, 4-24  
ippiCopyWrapBorder, 4-20  
ippiCountInRange, 11-30  
ippiCountZeros8x8, 16-161  
ippiCreateMapCameraUndistort, 14-75  
ippiCrossCorrFull\_Norm, 11-90  
ippiCrossCorrFull\_NormLevel, 11-96  
ippiCrossCorrSame\_Norm, 11-92

ippiCrossCorrValid\_Norm, 11-94  
ippiCrossCorrValid\_NormLevel, 11-100  
ippiDCT2x4x8Frw, 16-160  
ippiDCT2x4x8Inv, 16-157  
ippiDCT8x8Fwd, 10-50  
ippiDCT8x8FwdLS, 10-54  
ippiDCT8x8Inv, 10-52  
ippiDCT8x8Inv\_AANTransposed\_16s\_C1R, 16-129  
ippiDCT8x8Inv\_AANTransposed\_16s\_P2C2R, 16-131  
ippiDCT8x8Inv\_AANTransposed\_16s8u\_C1R, 16-130  
ippiDCT8x8Inv\_AANTransposed\_16s8u\_P2C2R,  
16-132  
ippiDCT8x8InvLSClip, 10-55  
ippiDCTFwd, 10-47  
ippiDCTFwdFree, 10-44  
ippiDCTFwdGetBufSize, 10-45  
ippiDCTFwdInitAlloc, 10-42  
ippiDCTInv, 10-49  
ippiDCTInvFree, 10-45  
ippiDCTInvGetBufSize, 10-46  
ippiDCTInvInitAlloc, 10-43  
ippiDCTQuantFwd8x8\_JPEG, 15-48  
ippiDCTQuantFwd8x8LS\_JPEG, 15-49  
ippiDCTQuantInv8x8\_JPEG, 15-50  
ippiDCTQuantInv8x8LS\_JPEG, 15-51  
ippiDecodeBlockCoef\_Inter\_H263, 16-287  
ippiDecodeBlockCoef\_Inter\_MPEG4, 16-226  
ippiDecodeBlockCoef\_Intra\_H263, 16-286  
ippiDecodeBlockCoef\_Intra\_MPEG4, 16-224  
ippiDecodeCAEInterH\_MPEG4, 16-231  
ippiDecodeCAEInterV\_MPEG4, 16-231  
ippiDecodeCAEIntraH\_MPEG4, 16-230  
ippiDecodeCAEIntraV\_MPEG4, 16-230  
ippiDecodeCAVLCChromaDcCoeffs\_H264, 16-337  
ippiDecodeCAVLCCoeffs\_H264, 16-335  
ippiDecodeCBProgrAttach\_JPEG2K, 15-159  
ippiDecodeCBProgrFree\_JPEG2K, 15-158  
ippiDecodeCBProgrGetStateSize, 15-156  
ippiDecodeCBProgrInit\_JPEG2K, 15-157  
ippiDecodeCBProgrInitAlloc\_JPEG2K, 15-157  
ippiDecodeCodeBlock\_JPEG2K, 15-154  
ippiDecodeCoeffsInter\_H261, 16-271  
ippiDecodeCoeffsInter\_MPEG4, 16-217  
ippiDecodeCoeffsInterRVLCBack\_MPEG4, 16-219  
ippiDecodeCoeffsIntra\_H261, 16-270  
ippiDecodeCoeffsIntra\_H263, 16-289  
ippiDecodeCoeffsIntra\_MPEG4, 16-215  
ippiDecodeCoeffsIntraRVLCBack\_MPEG4, 16-216  
ippiDecodeDCIntra\_H263, 16-288  
ippiDecodeDCIntra\_MPEG4, 16-214  
ippiDecodeExpGolombOne\_H264, 16-338  
ippiDecodeGetBufSize\_JPEG2K, 15-154  
ippiDecodeHuffman8x8\_ACFirst\_JPEG, 15-104  
ippiDecodeHuffman8x8\_ACRrefine\_JPEG, 15-105  
ippiDecodeHuffman8x8\_DCFfirst\_JPEG, 15-101  
ippiDecodeHuffman8x8\_DCRefine\_JPEG, 15-103  
ippiDecodeHuffman8x8\_Direct\_JPEG, 15-100  
ippiDecodeHuffman8x8\_JPEG, 15-98  
ippiDecodeHuffmanOne, 16-20  
ippiDecodeHuffmanOne\_JPEG, 15-113  
ippiDecodeHuffmanPair, 16-21  
ippiDecodeHuffmanSpecFree\_JPEG, 15-95  
ippiDecodeHuffmanSpecGetBufSize\_JPEG, 15-92  
ippiDecodeHuffmanSpecInit\_JPEG, 15-93  
ippiDecodeHuffmanSpecInitAlloc\_JPEG, 15-94  
ippiDecodeHuffmanStateFree\_JPEG, 15-98  
ippiDecodeHuffmanStateGetBufSize\_JPEG, 15-96  
ippiDecodeHuffmanStateInit\_JPEG, 15-96  
ippiDecodeHuffmanStateInitAlloc\_JPEG, 15-97  
ippiDecodeMV\_BVOP\_Backward\_MPEG4, 16-196  
ippiDecodeMV\_BVOP\_Direct\_MPEG4, 16-198  
ippiDecodeMV\_BVOP\_DirectSkip\_MPEG4, 16-200  
ippiDecodeMV\_BVOP\_Forward\_MPEG4, 16-195  
ippiDecodeMV\_BVOP\_Interpolate\_MPEG4, 16-197  
ippiDecodeMVS\_MPEG4, 16-233  
ippiDecodePadMV\_PVOP\_MPEG4, 16-193  
ippiDecodeVLCZigzag\_Inter\_MPEG4, 16-223  
ippiDecodeVLCZigzag\_IntraACVLC\_MPEG4, 16-221

---

ippiDecodeVLCZigzag_IntraDCVLC_MPEG4, 16-221	ippiEncodeFree_JPEG2K, 15-146
ippiDeconvFFT, 9-69	ippiEncodeGetDist_JPEG2K, 15-153
ippiDeconvFFTFree, 9-68	ippiEncodeGetRate_JPEG2K, 15-152
ippiDeconvFFTInitAlloc, 9-67	ippiEncodeGetTermPassLen_JPEG2K, 15-151
ippiDeconvLR, 9-71	ippiEncodeHuffman8x8_ACFirst_JPEG, 15-90
ippiDeconvLRFree, 9-71	ippiEncodeHuffman8x8_ACRrefine_JPEG, 15-91
ippiDeconvLRInitAlloc, 9-70	ippiEncodeHuffman8x8_DCFirst_JPEG, 15-87
ippiDeinterlaceFilterTriangle, 16-109	ippiEncodeHuffman8x8_DCRrefine_JPEG, 15-88
ippiDequantTransformResidual_H264, 16-341	ippiEncodeHuffman8x8_Direct_JPEG, 15-82
ippiDequantTransformResidual_SISP_H264, 16-346	ippiEncodeHuffman8x8_JPEG, 15-81
ippiDequantTransformResidualAndAdd_H264, 16-343	ippiEncodeHuffmanOne_JPEG, 15-112
ippiDFTFree, 10-17	ippiEncodeHuffmanRawTableInit_JPEG, 15-74
ippiDFTFwd, 10-19	ippiEncodeHuffmanSpecFree_JPEG, 15-78
ippiDFTGetBufSize, 10-18	ippiEncodeHuffmanSpecGetBufSize_JPEG, 15-75
ippiDFTInitAlloc, 10-16	ippiEncodeHuffmanSpecInit_JPEG, 15-76
ippiDFTInv, 10-22	ippiEncodeHuffmanSpecInitAlloc_JPEG, 15-77
ippiDiffPredFirstRow_JPEG, 15-107	ippiEncodeHuffmanStateFree_JPEG, 15-80
ippiDiffPredRow_JPEG, 15-108	ippiEncodeHuffmanStateGetBufSize_JPEG, 15-78
ippiDilate, 8-11	ippiEncodeHuffmanStateInit_JPEG, 15-79
ippiDilate3x3, 8-7	ippiEncodeHuffmanStateInitAlloc_JPEG, 15-80
ippiDilateBorderReplicate, 8-19	ippiEncodeInitAlloc_JPEG2K, 15-145
ippiDistanceTransform, 14-16	ippiEncodeLoadCodeBlock_JPEG2K, 15-146
ippiDiv, 5-34	ippiEncodeMV_MPEG4, 16-264
ippiDivC, 5-37	ippiEncodeStoreBits_JPEG2K, 15-149
ippiDownsampleFour_H263, 16-301	ippiEncodeTableInitAlloc, 16-140
ippiDownsampleFour16x16_H263, 16-320	ippiEncodeVLCZigzag_Inter_MPEG4, 16-259
ippiDup, 4-27	ippiEncodeVLCZigzag_IntraACVLC_MPEG4, 16-258
ippiEigenValsVecs, 14-10	ippiEncodeVLCZigzag_IntraDCVLC_MPEG4, 16-258
ippiEigenValsVecsGetBufferSize, 14-9	ippiErode, 8-13
ippiEncodeChromaDcCoeffsCAVLC_H264_16s, 16-429	ippiErode3x3, 8-9
ippiEncodeCoeffsCAVLC_H264_16s, 16-425	ippiErodeBorderReplicate, 8-21
ippiEncodeCoeffsInter_H261, 16-278	ippiExp, 5-52
ippiEncodeCoeffsInter_H263, 16-316	ippiExpandFrame_H263, 16-297
ippiEncodeCoeffsInter_MPEG4, 16-257	ippiExpandPlane_H264, 16-353
ippiEncodeCoeffsIntra_H261, 16-277	ippiFFTFree, 10-7
ippiEncodeCoeffsIntra_H263, 16-315	ippiFFTFwd, 10-9
ippiEncodeCoeffsIntra_MPEG4, 16-256	ippiFFTGetBufSize, 10-8
ippiEncodeDCIntra_H263, 16-314	ippiFFTInitAlloc, 10-6
ippiEncodeDCIntra_MPEG4, 16-255	ippiFFTInv, 10-14



ippiFilter, 9-31  
ippiFilter32f, 9-34  
ippiFilter8x8\_H261, 16-279  
ippiFilterBlockBoundaryHorEdge\_H263, 16-305  
ippiFilterBlockBoundaryVerEdge\_H263, 16-305  
ippiFilterBox, 9-8  
ippiFilterColumn, 9-36  
ippiFilterColumn32f, 9-38  
ippiFilterColumnPipeline, 9-52  
ippiFilterColumnPipeline\_Low, 9-54  
ippiFilterColumnPipelineGetBufferSize, 9-45  
ippiFilterColumnPipelineGetBufferSize\_Low, 9-46  
ippiFilterDeblocking16x16\_HorEdge\_H263, 16-308  
ippiFilterDeblocking16x16\_VerEdge\_H263, 16-308  
ippiFilterDeblocking8x8HorEdge\_H263, 16-306  
ippiFilterDeblocking8x8HorEdge\_MPEG4, 16-227  
ippiFilterDeblocking8x8VerEdge\_H263, 16-306  
ippiFilterDeblocking8x8VerEdge\_MPEG4, 16-227  
ippiFilterDeblockingChroma\_HorEdge\_H264, 16-413  
ippiFilterDeblockingChroma\_VerEdge\_H264, 16-409  
ippiFilterDeblockingChroma\_VerEdge\_MBAFF\_H264, 16-412  
ippiFilterDeblockingLuma\_HorEdge\_H264, 16-407  
ippiFilterDeblockingLuma\_VerEdge\_H264, 16-404  
ippiFilterDeblockingLuma\_VerEdge\_MBAFF\_H264, 16-406  
ippiFilterDeringingSmooth8x8\_MPEG4, 16-229  
ippiFilterDeringingThreshold\_MPEG4, 16-228  
ippiFilterGauss, 9-92  
ippiFilterHipass, 9-94  
ippiFilterLaplace, 9-91  
ippiFilterLaplacianBorder, 9-122  
ippiFilterLowpass, 9-95  
ippiFilterLowpassBorder, 9-124  
ippiFilterLowpassGetBufferSize, 9-107  
ippiFilterMax, 9-15  
ippiFilterMaxBorderReplicate, 9-20  
ippiFilterMaxGetBufferSize, 9-17  
ippiFilterMedian, 9-23  
ippiFilterMedianColor, 9-29  
ippiFilterMedianCross, 9-28  
ippiFilterMedianHoriz, 9-25  
ippiFilterMedianVert, 9-26  
ippiFilterMin, 9-12  
ippiFilterMinBorderReplicate, 9-18  
ippiFilterMinGetBufferSize, 9-16  
ippiFilterPrewittHoriz, 9-74  
ippiFilterPrewittVert, 9-75  
ippiFilterRobertsDown, 9-88  
ippiFilterRobertsUp, 9-89  
ippiFilterRow, 9-39  
ippiFilterRow32f, 9-42  
ippiFilterRowBorderPipeline, 9-47  
ippiFilterRowBorderPipeline\_Low, 9-50  
ippiFilterRowBorderPipelineGetBufferSize, 9-43  
ippiFilterRowBorderPipelineGetBufferSize\_Low, 9-44  
ippiFilterScharrHoriz, 9-78  
ippiFilterScharrHorizBorder, 9-109  
ippiFilterScharrHorizGetBufferSize, 9-99  
ippiFilterScharrVert, 9-79  
ippiFilterScharrVertBorder, 9-110  
ippiFilterScharrVertGetBufferSize, 9-100  
ippiFilterSharpen, 9-97  
ippiFilterSobelCross, 9-87  
ippiFilterSobelCrossBorder, 9-120  
ippiFilterSobelCrossGetBufferSize, 9-105  
ippiFilterSobelHoriz, 9-80, 9-81  
ippiFilterSobelHorizBorder, 9-112  
ippiFilterSobelHorizGetBufferSize, 9-101  
ippiFilterSobelHorizMask, 9-81  
ippiFilterSobelHorizSecond, 9-84  
ippiFilterSobelHorizSecondBorder, 9-116  
ippiFilterSobelHorizSecondGetBufferSize, 9-103  
ippiFilterSobelNegVertBorder, 9-114  
ippiFilterSobelNegVertGetBufferSize, 9-102  
ippiFilterSobelVert, 9-82  
ippiFilterSobelVertBorder, 9-114  
ippiFilterSobelVertGetBufferSize, 9-102

---

ippiFilterSobelVertMask, 9-82	ippiGetHuffmanStatistics8x8_ACRrefine_JPEG, 15-86
ippiFilterSobelVertSecond, 9-85	ippiGetHuffmanStatistics8x8_DCFirst_JPEG, 15-84
ippiFilterSobelVertSecondBorder, 9-118	ippiGetHuffmanStatistics8x8_JPEG, 15-83
ippiFilterSobelVertSecondGetBufferSize, 9-104	ippiGetHuffmanStatisticsOne_JPEG, 15-111
ippiFilterWiener, 9-58	ippiGetHuMoments, 11-52
ippiFilterWienerGetBufferSize, 9-57	ippiGetLibVersion, 3-2
ippiFindMVPred_MPEG4, 16-263	ippiGetNormalizedCentralMoment, 11-51
ippiFloodFill_Grad, 14-27	ippiGetNormalizedSpatialMoment, 11-48
ippiFloodFill_Range, 14-29	ippiGetPerspectiveBound, 12-63
ippiFloodFillGetSize, 14-23	ippiGetPerspectiveQuad, 12-62
ippiFloodFillGetSize_Grad, 14-24	ippiGetPerspectiveTransform, 12-64
ippiFrameFieldSAD16x16, 16-94	ippiGetPyramidDownROI, 14-52
ippiFree, 3-7	ippiGetResizeFract, 12-16
ippiFreeHuffmanTable_DV, 16-156	ippiGetRotateBound, 12-33
ippiGammaFwd, 6-183	ippiGetRotateQuad, 12-32
ippiGammaInv, 6-186	ippiGetRotateShift, 12-30
ippiGenScaleLevel8x8_H264, 16-424	ippiGetShearBound, 12-40
ippiGenSobelKernel, 9-108	ippiGetShearQuad, 12-39
ippiGetAffineBound, 12-52	ippiGetSpatialMoment, 11-47
ippiGetAffineQuad, 12-51	ippiGradientColorToGray, 14-21
ippiGetAffineTransform, 12-53	ippiGrayDilateBorder, 8-39, 8-40
ippiGetBilinearBound, 12-76	ippiHaarClassifierFree, 14-65
ippiGetBilinearQuad, 12-75	ippiHaarClassifierInitAlloc, 14-62
ippiGetBilinearTransform, 12-77	ippiHistogramEven, 11-27
ippiGetCentralMoment, 11-50	ippiHistogramRange, 11-24
ippiGetDiff16x16, 16-60	ippiHLSToBGR, 6-110
ippiGetDiff16x16B, 16-67	ippiHLSToRGB, 6-107
ippiGetDiff16x8, 16-61	ippiHSVToRGB, 6-113
ippiGetDiff16x8B, 16-69	ippiHuffmanDecodeSegment_DV, 16-154
ippiGetDiff4x4, 16-66	ippiHuffmanRunLevelTableInitAlloc, 16-19
ippiGetDiff8x16, 16-64	ippiHuffmanTableFree, 16-22
ippiGetDiff8x16B, 16-71	ippiHuffmanTableInitAlloc, 16-18
ippiGetDiff8x4, 16-65	ippiICTFwd_JPEG2K, 15-167
ippiGetDiff8x4B, 16-72	ippiICTInv_JPEG2K, 15-169
ippiGetDiff8x8, 16-62	ippiImageJaehne, 4-35
ippiGetDiff8x8B, 16-70	ippiImageRamp, 4-37
ippiGetDistanceTransformMask, 14-19	ippiInitAllocHuffmanTable_DV, 16-153
ippiGetHaarClassifierSize, 14-65	ippiIntegral, 11-7
ippiGetHuffmanStatistics8x8_ACFirst_JPEG, 15-85	ippiInterpolateAverage16x16, 16-53

ippiInterpolateAverage16x8, 16-53  
ippiInterpolateAverage8x4, 16-53  
ippiInterpolateAverage8x8, 16-53  
ippiInterpolateBlock\_H264, 16-367  
ippiInterpolateChroma\_H264, 16-361  
ippiInterpolateChromaBottom\_H264, 16-365  
ippiInterpolateChromaTop\_H264, 16-363  
ippiInterpolateLuma\_H264, 16-354  
ippiInterpolateLumaBottom\_H264, 16-358  
ippiInterpolateLumaTop\_H264, 16-356  
ippiJoin422LS\_MCU, 15-71  
ippiLabToBGR, 6-101  
ippiLFilterLaplacianGetBufferSize, 9-106  
ippiLimitMVToRect\_MPEG4, 16-201  
ippiLn, 5-49  
ippiLShiftC, 5-69  
ippiLUT, 6-125  
ippiLUT\_Cubic, 6-132  
ippiLUT\_Linear, 6-129  
ippiLUTPalette, 6-137  
ippiLUVToRGB, 6-97  
ippiMagnitude, 10-30  
ippiMagnitudePack, 10-31  
ippiMalloc, 3-5  
ippiMax, 11-34  
ippiMaxIndx, 11-36  
ippiMC16x16, 16-24  
ippiMC16x16B, 16-36  
ippiMC16x4B, 16-48  
ippiMC16x8, 16-25  
ippiMC16x8B, 16-38  
ippiMC16x8BUV, 16-50  
ippiMC16x8UV, 16-35  
ippiMC2x2, 16-33  
ippiMC2x2B, 16-47  
ippiMC2x4, 16-31  
ippiMC2x4B, 16-45  
ippiMC4x2, 16-32  
ippiMC4x2B, 16-46  
ippiMC4x4, 16-30  
ippiMC4x4B, 16-44  
ippiMC4x8, 16-29  
ippiMC4x8B, 16-42  
ippiMC8x16, 16-26  
ippiMC8x16B, 16-39  
ippiMC8x4, 16-28, 16-34  
ippiMC8x4B, 16-41  
ippiMC8x8, 16-27  
ippiMC8x8B, 16-40  
ippiMean, 11-15  
ippiMean\_StdDev, 11-18  
ippiMeanAbsDev16x16, 16-86  
ippiMin, 11-31  
ippiMinEigenVal, 14-14  
ippiMinEigenValGetBufferSize, 14-13  
ippiMinIndx, 11-33  
ippiMinMax, 11-37  
ippiMinMaxIndx, 11-39  
ippiMirror, 12-21  
ippiMomentFree, 11-43  
ippiMomentGetStateSize, 11-44  
ippiMomentInit, 11-45  
ippiMomentInitAlloc, 11-42  
ippiMoments, 11-45  
ippiMorpAdvGetSize, 8-26  
ippiMorpGrayGetSize, 8-38  
ippiMorphAdvFree, 8-24  
ippiMorphAdvInit, 8-24, 8-37  
ippiMorphAdvInitAlloc, 8-22, 8-35  
ippiMorphBlackhatBorder, 8-31  
ippiMorphGradientBorder, 8-33  
ippiMorphGrayFree, 8-36  
ippiMorphologyFree, 8-16  
ippiMorphologyGetSize, 8-18  
ippiMorphologyInit, 8-17  
ippiMorphologyInitAlloc, 8-15  
ippiMorphOpenBorder, 8-27, 8-28  
ippiMorphReconstructErode, 8-46

---

ippiMorphReconstructGetBufferSize, 8-42	ippiPredictIntra_16x16_H264, 16-350
ippiMorphTophatBorder, 8-30	ippiPredictIntra_4x4_H264, 16-348
ippiMotionEstimation_16x16_MVFAST, 16-248	ippiPredictIntraChroma8x8_H264, 16-351
ippiMotionEstimation_16x16_SEA, 16-245	ippiPredictReconCoeffIntra_MPEG4, 16-202
ippiMul, 5-16	ippiPutIntraBlock, 16-141
ippiMulC, 5-19	ippiPutNonIntraBlock, 16-142
ippiMulCScale, 5-25	ippiPyramidFree, 14-46
ippiMulPack, 10-25	ippiPyramidInitAlloc, 14-45
ippiMulPackConj, 10-28	ippiPyramidLayerDown, 14-54
ippiMulScale, 5-23	ippiPyramidLayerDownFree, 14-49
ippiNorm_Inf, 11-54	ippiPyramidLayerUp, 14-56
ippiNorm_L1, 11-56	ippiPyramidLayerUpFree, 14-51
ippiNorm_L2, 11-60	ippiPyramidLayerUpInitAlloc, 14-49
ippiNormDiff_Inf, 11-62	ippiPyrDown, 14-42
ippiNormDiff_L1, 11-65	ippiPyrDownGetBufSize, 14-40
ippiNormDiff_L2, 11-68	ippiPyrUp, 14-43
ippiNormRel_Inf, 11-71	ippiPyrUpGetBufSize, 14-41
ippiNormRel_L1, 11-73	ippiQualityIndex, 11-80
ippiNormRel_L2, 11-76	ippiQuant_MPEG2, 16-139
ippiNot, 5-59	ippiQuantFwd8x8_JPEG, 15-45
ippiOBMC16x16HP_MPEG4, 16-185	ippiQuantFwdRawTableInit_JPEG, 15-43
ippiOBMC8x8HP_MPEG4, 16-185	ippiQuantFwdTableInit_JPEG, 15-44
ippiOBMC8x8QP_MPEG4, 16-185	ippiQuantInter_H263, 16-319
ippiOpticalFlowPyrLKFree, 14-36	ippiQuantInter_MPEG4, 16-253
ippiOpticalFlowPyrLKInitAlloc, 14-35	ippiQuantInterGetSize_MPEG4, 16-252
ippiOr, 5-61	ippiQuantInterInit_MPEG4, 16-251
ippiOrC, 5-62	ippiQuantIntra_H263, 16-318
ippiPackToCplxExtend, 10-36	ippiQuantIntra_MPEG2, 16-138
ippiPadCurrent_16x16_MPEG4, 16-207	ippiQuantIntra_MPEG4, 16-253
ippiPadCurrent_8x8_MPEG4, 16-207	ippiQuantIntraGetSize_MPEG4, 16-252
ippiPadMBGray_MPEG4, 16-206	ippiQuantIntraInit_MPEG4, 16-251
ippiPadMBHorizontal_MPEG4, 16-204	ippiQuantInv_DV, 16-156
ippiPadMBOpaque_MPEG4, 16-237	ippiQuantInv_MPEG2, 16-128
ippiPadMBPartial_MPEG4, 16-234	ippiQuantInv8x8_JPEG, 15-46
ippiPadMBTransparent_MPEG4, 16-235	ippiQuantInvInter_H263, 16-294
ippiPadMBVertical_MPEG4, 16-205	ippiQuantInvInter_MPEG4, 16-212
ippiPadMV_MPEG4, 16-209	ippiQuantInvInterGetSize_MPEG4, 16-211
ippiPhase, 10-32	ippiQuantInvInterInit_MPEG4, 16-210
ippiPhasePack, 10-34	ippiQuantInvIntra_H263, 16-292

ippiQuantInvIntra\_MPEG2, 16-127  
ippiQuantInvIntra\_MPEG4, 16-212  
ippiQuantInvIntraGetSize\_MPEG4, 16-211  
ippiQuantInvIntraInit\_MPEG4, 16-210  
ippiQuantInvTableInit\_JPEG, 15-46  
ippiQuantLuma8x8\_H264, 16-423  
ippiQuantLuma8x8Inv\_H264, 16-430  
ippiRCTFwd\_JPEG2K, 15-164  
ippiRCTInv\_JPEG2K, 15-166  
ippiReconstructChromaInter4x4MB\_H264, 16-380  
ippiReconstructChromaInterMB\_H264, 16-374  
ippiReconstructChromaIntra4x4MB\_H264, 16-384  
ippiReconstructChromaIntraHalves4x4MB\_H264, 16-382  
ippiReconstructChromaIntraHalvesMB\_H264, 16-376  
ippiReconstructChromaIntraMB\_H264, 16-378  
ippiReconstructCoeffsInter\_H261, 16-274  
ippiReconstructCoeffsInter\_H263, 16-311  
ippiReconstructCoeffsInter\_MPEG4, 16-220  
ippiReconstructCoeffsIntra\_H261, 16-272  
ippiReconstructCoeffsIntra\_H263, 16-309  
ippiReconstructDCTBlock\_MPEG1, 16-119  
ippiReconstructDCTBlock\_MPEG2, 16-122  
ippiReconstructDCTBlockIntra\_MPEG1, 16-120  
ippiReconstructDCTBlockIntra\_MPEG2, 16-125  
ippiReconstructLumaInter4x4MB\_H264, 16-391  
ippiReconstructLumaInter8x8MB\_H264, 16-395  
ippiReconstructLumaIntera4x4MB\_H264, 16-394  
ippiReconstructLumaInteraHalf4x4MB\_H264, 16-392  
ippiReconstructLumaInterMB\_H264, 16-386  
ippiReconstructLumaIntra16x16MB\_H264, 16-400, 16-402  
ippiReconstructLumaIntra8x8MB\_H264, 16-398  
ippiReconstructLumaIntraHalf8x8MB\_H264, 16-397  
ippiReconstructLumaIntraHalfMB\_H264, 16-387  
ippiReconstructLumaIntraMB\_H264, 16-389  
ippiReconstructPredFirstRow\_JPEG, 15-109  
ippiReconstructPredRow, 15-110  
ippiRectStsDev, 11-20  
ippiReduceBits, 6-138  
ippiRemap, 12-23  
ippiResample\_H263, 16-298  
ippiResize, 12-5  
ippiResizeCCRotate, 16-106  
ippiResizeCenter, 12-8  
ippiResizeShift, 12-17  
ippiResizeSqrPixel, 12-12  
ippiResizeSqrPixelGetBufSize, 12-15  
ippiResizeYUV422, 12-20  
ippiRGB555ToYCbCr\_JPEG, 15-8  
ippiRGB565ToYCbCr\_JPEG, 15-8  
ippiRGBToCbYCr422, 6-65  
ippiRGBToCbYCr422Gamma, 6-65  
ippiRGBToGray, 6-123  
ippiRGBToHLS, 6-105  
ippiRGBToHSV, 6-111  
ippiRGBToLUV, 6-95  
ippiRGBToXYZ, 6-93  
ippiRGBToY\_JPEG, 15-4  
ippiRGBToYCbCr, 6-48  
ippiRGBToYCbCr\_JPEG, 15-6  
ippiRGBToYCbCr411LS\_MCU, 15-21  
ippiRGBToYCbCr420, 6-75  
ippiRGBToYCbCr422, 6-58  
ippiRGBToYCbCr422LS\_MCU, 15-20  
ippiRGBToYCbCr444LS\_MCU, 15-19  
ippiRGBToYCC, 6-102  
ippiRGBToYUV, 6-31  
ippiRGBToYUV420, 6-38  
ippiRGBToYUV422, 6-34  
ippiRotate, 12-26  
ippiRotateCenter, 12-34  
ippiRShiftC, 5-72  
ippiSAD16x16, 16-88  
ippiSAD16x16Blocks4x4, 16-93  
ippiSAD16x16Blocks8x8, 16-92  
ippiSAD16x8, 16-89  
ippiSAD4x4, 16-91

---

ippiSAD8x8, 16-90	ippiSubSAD8x8, 16-74
ippiSampleDown411LS_MCU, 15-66	ippiSum, 11-5
ippiSampleDown422LS_MCU, 15-65	ippiSumNorm_VOP_MPEG4, 16-242
ippiSampleDown444LS_MCU, 15-64	ippiSumsDiff16x16Blocks4x4, 16-96
ippiSampleDownH2V1_JPEG, 15-55	ippiSumsDiff16x16Blocks4x4_8u16s_C1, 16-96
ippiSampleDownH2V2_JPEG, 15-56	ippiSumsDiff8x8Blocks4x4, 16-98
ippiSampleDownRowH2V1_Box_JPEG, 15-57	ippiSumsDiff8x8Blocks4x4_8u16s_C1, 16-98
ippiSampleDownRowH2V2_Box_JPEG, 15-58	ippiSumWindowColumn, 9-11
ippiSampleLine, 4-39	ippiSumWindowRow, 9-10
ippiSampleUp411LS_MCU, 15-69	ippiSwapChannels, 4-30
ippiSampleUp444LS_MCU, 15-67	ippiThreshold, 7-2
ippiSampleUpH2V1_JPEG, 15-59	ippiThreshold_GT, 7-5
ippiSampleUpH2V2_JPEG, 15-61	ippiThreshold_GTVal, 7-11
ippiSampleUpRowH2V1_Triangle_JPEG, 15-62	ippiThreshold_LT, 7-7
ippiSampleUpRowH2V2_Triangle_JPEG, 15-63	ippiThreshold_LTVal, 7-13
ippiSBGRToYCoCg, 6-115	ippiThreshold_LTValGTVal, 7-16
ippiSBGRToYCoCg_Rev, 6-119	ippiThreshold_Val, 7-9
ippiScale, 4-6	ippiTiltedHaarClassifierInitAlloc, 14-63
ippiScanFwd, 16-101	ippiTiltedIntegral, 11-11
ippiScanInv, 16-99	ippiTiltedRectStdDev, 11-22
ippiSet, 4-9	ippiTiltedSqrIntegral, 11-13
ippiShear, 12-36	ippiTransformDequantChromaDC_H264, 16-340
ippiSpatialInterpolation_H263, 16-303	ippiTransformDequantChromaDC_SISP_H264, 16-347
ippiSplit422LS_MCU, 15-70	ippiTransformDequantLumaDC_H264, 16-339
ippiSqr, 5-44	ippiTransformLuma8x8Fwd_H264, 16-422
ippiSqrDiff16x16, 16-76	ippiTransformLuma8x8InvAddPred_H264, 16-431
ippiSqrDiff16x16B, 16-77	ippiTransformPrediction_H264, 16-345
ippiSqrDistanceFull_Norm, 11-84	ippiTransformQuantChromaDC_H264, 16-417
ippiSqrDistanceSame_Norm, 11-86	ippiTransformQuantLumaDC_H264, 16-419
ippiSqrDistanceValid_Norm, 11-88	ippiTransformQuantResidual_H264, 16-421
ippiSqrIntegral, 11-9	ippiTranspose, 4-28
ippiSqrt, 5-46	ippiTransRecBlockCoef_inter_MPEG4, 16-260
ippiSSD4x4, 16-79	ippiTransRecBlockCoef_intra_MPEG4, 16-261
ippiSSD8x8, 16-78	ippiUndistortGetSize, 14-72
ippiSub, 5-27	ippiUndistortRadial, 14-73
ippiSub128_JPEG, 15-52	ippiUniDirWeightBlock_H264, 16-369
ippiSub16x16, 16-73, 16-74	ippiUpdateMotionHistory, 14-33
ippiSub8x8, 16-73	ippiUpsampleFour_H263, 16-300
ippiSubC, 5-30	ippiUpsampleFour8x8_H263, 16-302

ippiVariance16x16, 16-85, 16-87  
ippiVarMean8x8\_8u32s, 16-80  
ippiVarMeanDiff16x16, 16-82  
ippiVarMeanDiff16x8, 16-83  
ippiWarpAffine, 12-41  
ippiWarpAffineBack, 12-45  
ippiWarpAffineQuad, 12-48  
ippiWarpBilinear, 12-66  
ippiWarpBilinearBack, 12-69  
ippiWarpBilinearQuad, 12-72  
ippiWarpChroma\_MPEG4, 16-190  
ippiWarpGetSize\_MPEG4, 16-188  
ippiWarpInit\_MPEG4, 16-187  
ippiWarpLuma\_MPEG4, 16-189  
ippiWarpPerspective, 12-54  
ippiWarpPerspectiveBack, 12-57  
ippiWarpPerspectiveQuad, 12-60  
ippiWeightedAverage\_H264, 16-368  
ippiWTFwd, 13-8  
ippiWTFwd\_B53\_JPEG2K, 15-136  
ippiWTFwdCol\_B53\_JPEG2K, 15-120  
ippiWTFwdCol\_D97\_JPEG2K, 15-129  
ippiWTFwdColLift\_B53\_JPEG2K, 15-121  
ippiWTFwdColLift\_D97\_JPEG2K, 15-130  
ippiWTFwdFree, 13-7  
ippiWTFwdGetBufSize, 13-7  
ippiWTFwdInitAlloc, 13-5  
ippiWTFwdRow\_B53\_JPEG2K, 15-117  
ippiWTFwdRow\_D97\_JPEG2K, 15-126  
ippiWTInv, 13-17  
ippiWTInv\_B53\_JPEG2K, 15-138  
ippiWTInv\_D97\_JPEG2K, 15-142  
ippiWTInvCol\_B53\_JPEG2K, 15-123  
ippiWTInvCol\_D97\_JPEG2K, 15-132  
ippiWTInvColLift\_B53\_JPEG2K, 15-124  
ippiWTInvColLift\_D97\_JPEG2K, 15-133  
ippiWTInvFree, 13-15  
ippiWTInvGetBufSize, 13-16  
ippiWTInvInitAlloc, 13-13  
ippiWTInvRow\_B53\_JPEG2K, 15-118  
ippiWTInvRow\_D97\_JPEG2K, 15-127  
ippiXor, 5-65  
ippiXorC, 5-67  
ippiXYZToRGB, 6-94  
ippiYCbCr411, 6-171  
ippiYCbCr411ToBGR, 6-89  
ippiYCbCr411ToBGR555, 6-92  
ippiYCbCr411ToBGR555LS\_MCU, 15-39  
ippiYCbCr411ToBGR565, 6-92  
ippiYCbCr411ToBGR565LS\_MCU, 15-39  
ippiYCbCr411ToBGR565LS\_MCU, 15-39  
ippiYCbCr411ToBGR565LS\_MCU, 15-38  
ippiYCbCr411ToRGBLS\_MCU, 15-33  
ippiYCbCr411ToYCbCr420, 6-175  
ippiYCbCr411ToYCbCr422, 6-173  
ippiYCbCr411ToYCrCb420, 6-177  
ippiYCbCr411ToYCrCb422, 6-174  
ippiYCbCr420, 6-156  
ippiYCbCr420ToBGR, 6-81  
ippiYCbCr420ToBGR444Dither, 6-85  
ippiYCbCr420ToBGR555Dither, 6-85  
ippiYCbCr420ToBGR565Dither, 6-85  
ippiYCbCr420ToCbYCr422, 6-161  
ippiYCbCr420ToRGB, 6-76  
ippiYCbCr420ToRGB444Dither, 6-79  
ippiYCbCr420ToRGB555Dither, 6-79  
ippiYCbCr420ToRGB565Dither, 6-79  
ippiYCbCr420ToYCbCr411, 6-165  
ippiYCbCr420ToYCbCr422, 6-157  
ippiYCbCr420ToYCbCr422\_Filter, 6-159  
ippiYCbCr420ToYCrCb420, 6-162  
ippiYCbCr420ToYCrCb420\_Filter, 6-163  
ippiYCbCr422, 6-141  
ippiYCbCr422ToBGR, 6-62  
ippiYCbCr422ToBGR444, 6-72  
ippiYCbCr422ToBGR444Dither, 6-74  
ippiYCbCr422ToBGR555, 6-72  
ippiYCbCr422ToBGR555Dither, 6-74  
ippiYCbCr422ToBGR555LS\_MCU, 15-37

---

ippiYCbCr422ToBGR565, 6-72	ippiYCbCrToRGB565_JPEG, 15-9
ippiYCbCr422ToBGR565Dither, 6-74	ippiYCbCrToRGB565Dither, 6-54
ippiYCbCr422ToBGR565LS_MCU, 15-37	ippiYCKK444ToCMYKLS_MCU, 15-40
ippiYCbCr422ToBGR565LS_MCU, 15-36	ippiYCKKToCMYK_JPEG, 15-16
ippiYCbCr422ToCbYCr422, 6-143	ippiYCKKToCMYK411LS_MCU, 15-42
ippiYCbCr422ToRGB, 6-59	ippiYCKKToCMYK422LS_MCU, 15-41
ippiYCbCr422ToRGB444, 6-69	ippiYCCToRGB, 6-104
ippiYCbCr422ToRGB444Dither, 6-71	ippiYCiCgToSBGR, 6-117
ippiYCbCr422ToRGB555, 6-69	ippiYCoCgToBGR, 6-116
ippiYCbCr422ToRGB555Dither, 6-71	ippiYCoCgToBGR_Rev, 6-120
ippiYCbCr422ToRGB565, 6-69	ippiYCoCgToSBGR_Rev, 6-121
ippiYCbCr422ToRGB565Dither, 6-71	ippiYCrCb411ToYCbCr422_5MBDV, 16-162
ippiYCbCr422ToRGBLS_MCU, 15-32	ippiYCrCb411ToYCbCr422_EdgeDV, 16-163
ippiYCbCr422ToYCbCr411, 6-147	ippiYCrCb411ToYCbCr422_ZoomOut2_5MBDV, 16-162
ippiYCbCr422ToYCbCr420, 6-144	ippiYCrCb411ToYCbCr422_ZoomOut2_EdgeDV, 16-163
ippiYCbCr422ToYCrCb420, 6-146	ippiYCrCb411ToYCbCr422_ZoomOut4_5MBDV, 16-162
ippiYCbCr422ToYCrCb422, 6-142	ippiYCrCb411ToYCbCr422_ZoomOut4_EdgeDV, 16-163
ippiYCbCr444ToBGR555LS_MCU, 15-35	ippiYCrCb411ToYCbCr422_ZoomOut8_5MBDV, 16-162
ippiYCbCr444ToBGR565LS_MCU, 15-35	ippiYCrCb411ToYCbCr422_ZoomOut8_EdgeDV, 16-163
ippiYCbCr444ToBGR565LS_MCU, 15-34	ippiYCrCb420ToCbYCr422, 6-168
ippiYCbCr444ToRGBLS_MCU, 15-31	ippiYCrCb420ToYCbCr411, 6-170
ippiYCbCrToBGR, 6-51	ippiYCrCb420ToYCbCr420, 6-169
ippiYCbCrToBGR_JPEG, 15-11	ippiYCrCb420ToYCbCr422, 6-166
ippiYCbCrToBGR444, 6-55	ippiYCrCb420ToYCbCr422_5MBDV, 16-164
ippiYCbCrToBGR444Dither, 6-57	ippiYCrCb420ToYCbCr422_Filter, 6-167
ippiYCbCrToBGR555, 6-55	ippiYCrCb420ToYCbCr422_ZoomOut2_5MBDV, 16-164
ippiYCbCrToBGR555_JPEG, 15-13	ippiYCrCb420ToYCbCr422_ZoomOut4_5MBDV, 16-164
ippiYCbCrToBGR555Dither, 6-57	ippiYCrCb420ToYCbCr422_ZoomOut8_5MBDV, 16-164
ippiYCbCrToBGR565, 6-55	ippiYCrCb422ToYCbCr411, 6-151
ippiYCbCrToBGR565_JPEG, 15-13	ippiYCrCb422ToYCbCr420, 6-150
ippiYCbCrToBGR565Dither, 6-57	ippiYCrCb422ToYCbCr422, 6-149
ippiYCbCrToRGB, 6-50	
ippiYCbCrToRGB_JPEG, 15-7	
ippiYCbCrToRGB444, 6-52	
ippiYCbCrToRGB444Dither, 6-54	
ippiYCbCrToRGB555, 6-52	
ippiYCbCrToRGB555_JPEG, 15-9	
ippiYCbCrToRGB555Dither, 6-54	
ippiYCbCrToRGB565, 6-52	



ippiYCrCb422ToYCbCr422\_5MBDV, 16-166  
ippiYCrCb422ToYCbCr422\_ZoomOut2\_5MBDV,  
16-166  
ippiYCrCb422ToYCbCr422\_ZoomOut4\_5MBDV,  
16-166  
ippiYCrCb422ToYCbCr422\_ZoomOut8\_5MBDV,  
16-166  
ippiYUV420ToBGR, 6-41  
ippiYUV420ToBGR444, 6-46  
ippiYUV420ToBGR444Dither, 6-47  
ippiYUV420ToBGR555, 6-46  
ippiYUV420ToBGR555Dither, 6-47  
ippiYUV420ToBGR565, 6-46  
ippiYUV420ToBGR565Dither, 6-47  
ippiYUV420ToRGB, 6-39  
ippiYUV420ToRGB444, 6-42  
ippiYUV420ToRGB444Dither, 6-43  
ippiYUV420ToRGB555, 6-42  
ippiYUV420ToRGB555Dither, 6-43  
ippiYUV420ToRGB565, 6-42  
ippiYUV420ToRGB565Dither, 6-43  
ippiYUV422ToRGB, 6-36  
ippiYUVToRGB, 6-33  
ippiZigzagFwd8x8, 4-40  
ippiZigzagInv8x8, 4-41  
ippiZigzagInvClassical\_Compact, 16-102  
ippiZigzagInvHorizontal\_Compact, 16-102  
ippiZigzagInvVertical\_Compact, 16-102  
ippjGetLibVersion, 15-2  
ippsApplyHaarClassifier, 14-67

## J

Join422LS\_MCU, 15-71  
JPEG Codec  
    color conversion functions, 15-3 thru 15-17  
    combined color conversion functions, 15-18 thru  
        15-42  
    combined quantization, DCT, and level shift  
        functions, 15-48 thru 15-51  
    entropy coding and decoding functions, 15-144 thru

15-156  
Huffman codec functions, 15-72 thru 15-106  
level shift functions, 15-52 thru 15-54  
library version, 15-2  
quantization functions, 15-43 thru 15-47  
sampling functions, 15-54 thru 15-62  
wavelet transform functions, 15-114 thru 15-143

## L

LabToBGR, 6-101  
level shift functions, 15-52  
LimitMVToRect\_MPEG4, 16-201  
lines padding, 2-22  
Ln, 5-49  
logical functions  
    bitwise AND, 5-55  
    bitwise NOT, 5-59  
    bitwise OR, 5-61  
    bitwise XOR, 5-65  
    left shift, 5-69  
    right shift, 5-72  
logical operations, 5-55  
LShiftC, 5-69  
LUT, 6-125  
LUT\_Cubic, 6-132  
LUT\_Linear, 6-129  
LUTPalette, 6-137  
LUVToRGB, 6-97

## M

macroblock, 16-145, 16-269  
macroblock types, 16-176  
Magnitude, 10-30  
MagnitudePack, 10-31  
Malloc, 3-5  
manual organization, 1-5  
mapping, of data range, 4-7  
mask, in median filtering, 9-22  
masked operation, 4-13  
Max, 11-34

---

MaxIndx, 11-36  
MC16x16, 16-24  
MC16x16B, 16-36  
MC16x4, 16-34  
MC16x4B, 16-48  
MC16x8, 16-25  
MC16x8B, 16-38  
MC16x8BUV, 16-50  
MC16x8UV, 16-35  
MC2x2, 16-33  
MC2x2B, 16-47  
MC2x4, 16-31  
MC2x4B, 16-45  
MC4x2, 16-32  
MC4x2B, 16-46  
MC4x4, 16-30  
MC4x4B, 16-44  
MC4x8, 16-29  
MC4x8B, 16-42  
MC8x16, 16-26  
MC8x16B, 16-39  
MC8x4, 16-28  
MC8x4B, 16-41  
MC8x8, 16-27  
MC8x8B, 16-40  
MCU, minimum coded units, 15-19  
Mean, 11-15  
Mean\_StdDev, 11-18  
MeanAbsDev16x16, 16-86  
median filters, 9-22  
Memory allocation functions, 3-5 thru 3-7  
memory allocation functions  
    ippiFree, 3-7  
    ippiMalloc, 3-5  
Min, 11-31  
MinEigenVal, 14-14  
MinEigenValGetBufferSize, 14-13  
MinIndx, 11-33  
MinMax, 11-37

MinMaxIndx, 11-39  
Mirror, 12-21  
MMX technology, 1-1  
MomentFree, 11-43  
MomentGetStateSize, 11-44  
MomentInit, 11-45  
MomentInitAlloc, 11-42  
Moments, 11-45  
moments, of an image, 11-41  
MorphAdvFree, 8-24  
MorphAdvGetSize, 8-26  
MorphAdvInit, 8-24  
MorphAdvInitAlloc, 8-22  
MorphBlackhatBorder, 8-31  
MorphCloseBorder, 8-28  
MorphGradientBorder, 8-33  
MorphGrayFree, 8-36  
MorphGrayGetSize, 8-38  
MorphGrayInit, 8-37  
MorphGrayInitAlloc, 8-35  
morphological functions  
    DilateBorderReplicate, 8-19  
    dilation, 8-7  
    ErodeBorderReplicate, 8-21  
    erosion, 8-9  
    GrayDilateBorder, 8-39, 8-40  
    MorphAdvFree, 8-24  
    MorphAdvGetSize, 8-26  
    MorphBlackhatBorder, 8-31  
    MorphCloseBorder, 8-28  
    MorphGradientBorder, 8-33  
    MorphGrayFree, 8-36  
    MorphGrayGetSize, 8-38  
    MorphologyFree, 8-16  
    MorphOpenBorder, 8-27  
    MorphReconstructErode, 8-46  
    MorphReconstructGetBufferSize, 8-42  
    MorphTophatBorder, 8-30  
MorphologyFree, 8-16  
MorphologyGetSize, 8-18  
MorphologyInit, 8-17  
MorphologyInitAlloc, 8-15

- MorphOpenBorder, 8-27
- MorphReconstructErode, 8-46
- MorphReconstructGetBufferSize, 8-42
- MorphTophatBorder, 8-30
- motion Analysis and object tracking, 14-31
- motion analysis and object tracking functions
  - OpticalFlowPyrLK, 14-36
  - OpticalFlowPyrLKFree, 14-36
  - OpticalFlowPyrLKInitAlloc, 14-35
  - UpdateMotionHistory, 14-33
- motion compensation adding functions, 16-54, 16-55, 16-56
- motion compensation averaging functions, 16-57
- motion compensation copying functions, 16-51, 16-52
- motion compensation interpolating functions, 16-53
- motion estimation subtracting functions, 16-73, 16-74
- motion vector buffers, 16-182
- motion vectors, in a macroblock, 16-177
- MotionEstimation\_16x16\_MVFAST, 16-248
- MotionEstimation\_16x16\_SEA, 16-245
- MPEG4 video decoder functions
  - CalcGlobalMV\_MPEG4, 16-191
  - ChangeSpriteBrightness\_MPEG4, 16-191
  - Copy16x16QP\_MPEG4, 16-184
  - Copy16x8QP\_MPEG4, 16-184
  - Copy8x8QP\_MPEG4, 16-184
  - DecodeBlockCoef\_Inter\_MPEG4, 16-226
  - DecodeBlockCoef\_Intra\_MPEG4, 16-224
  - DecodeCAEInterH\_MPEG4, 16-231
  - DecodeCAEInterV\_MPEG4, 16-231
  - DecodeCAEIntraH\_MPEG4, 16-230
  - DecodeCAEIntraV\_MPEG4, 16-230
  - DecodeCoeffsInter\_MPEG4, 16-217
  - DecodeCoeffsInterRVLCBack\_MPEG4, 16-219
  - DecodeCoeffsIntra\_MPEG4, 16-215
  - DecodeCoeffsIntraRVLCBack\_MPEG4, 16-216
  - DecodeDCIntra\_MPEG4, 16-214
  - DecodeMV\_BVOP\_Backward\_MPEG4, 16-196
  - DecodeMV\_BVOP\_Direct\_MPEG4, 16-198
  - DecodeMV\_BVOP\_DirectSkip\_MPEG4, 16-200
  - DecodeMV\_BVOP\_Forward\_MPEG4, 16-195
  - DecodeMV\_BVOP\_Interpolate\_MPEG4, 16-197
  - DecodeMVS\_MPEG4, 16-233
  - DecodePadMV\_PVOP\_MPEG4, 16-193
  - DecodeVLCZigzag\_Inter\_MPEG4, 16-223
  - DecodeVLCZigzag\_IntraACVLC\_MPEG4, 16-221
  - DecodeVLCZigzag\_IntraDCVLC\_MPEG4, 16-221
  - EncodeCoeffsInter\_MPEG4, 16-257
  - EncodeCoeffsIntra\_MPEG4, 16-256
  - EncodeDCIntra\_MPEG4, 16-255
  - FilterDeblocking8x8HorEdge\_MPEG4, 16-227
  - FilterDeblocking8x8VerEdge\_MPEG4, 16-227
  - FilterDeringingSmooth8x8\_MPEG4, 16-229
  - FilterDeringingThreshold\_MPEG4, 16-228
  - LimitMVToRect\_MPEG4, 16-201
  - OBMC16x16HP\_MPEG4, 16-185
  - OBMC8x8HP\_MPEG4, 16-185
  - OBMC8x8QP\_MPEG4, 16-185
  - PadCurrent\_16x16\_MPEG4, 16-207
  - PadCurrent\_8x8\_MPEG4, 16-207
  - PadMBGray\_MPEG4, 16-206
  - PadMBHorizontal\_MPEG4, 16-204
  - PadMBOpaque\_MPEG4, 16-237
  - PadMBPartial\_MPEG4, 16-234
  - PadMBTransparent\_MPEG4, 16-235
  - PadMBVertical\_MPEG4, 16-205
  - PadMV\_MPEG4, 16-209
  - PredictReconCoefIntra\_MPEG4, 16-202
  - QuantInter\_MPEG4, 16-253
  - QuantInterGetSize\_MPEG4, 16-252
  - QuantInterInit\_MPEG4, 16-251
  - QuantIntra\_MPEG4, 16-253
  - QuantIntraGetSize\_MPEG4, 16-252
  - QuantIntraInit\_MPEG4, 16-251
  - QuantInvInter\_MPEG4, 16-212
  - QuantInvInterGetSize\_MPEG4, 16-211
  - QuantInvInterInit\_MPEG4, 16-210
  - QuantInvIntra\_MPEG4, 16-212
  - QuantInvIntraGetSize\_MPEG4, 16-211
  - QuantInvIntraInit\_MPEG4, 16-210
  - ReconstructCoeffsInter\_MPEG4, 16-220
  - WarpChroma\_MPEG4, 16-190
  - WarpGetSize\_MPEG4, 16-188
  - WarpInit\_MPEG4, 16-187
  - WarpLuma\_MPEG4, 16-189
- MPEG-4 video encoder
  - MV FAST buffer, 16-239
  - rate control, 16-240
- Mul, 5-16

---

MulC, 5-19  
MulCScale, 5-25  
MulPack, 10-25  
MulPackConj, 10-28  
MulScale, 5-23

## N

naming conventions, 1-7  
neighborhood operations, 2-20  
Norm\_Inf, 11-54  
Norm\_L1, 11-56  
Norm\_L2, 11-60  
NormDiff\_Inf, 11-62  
NormDiff\_L1, 11-65  
NormDiff\_L2, 11-68  
NormRel\_Inf, 11-71  
NormRel\_L1, 11-73  
NormRel\_L2, 11-76  
norms, of an image, 11-54  
Not, 5-59  
notational conventions, 1-7

## O

object detection functions  
    HaarClassifierFree, 14-65  
    HaarClassifierInitAlloc, 14-62  
    TiltedHaarClassifierInitAlloc, 14-63  
OBMC16x16HP\_MPEG4, 16-185  
OBMC8x8HP\_MPEG4, 16-185  
OBMC8x8QP\_MPEG4, 16-185  
opening, 8-2  
operands, 2-6  
operation model, of image processing, 2-20  
optical flow, 14-34  
OpticalFlowPyrLK, 14-36  
OpticalFlowPyrLKFree, 14-36  
OpticalFlowPyrLKInitAlloc, 14-35  
Or, 5-61

OrC, 5-62

## P

PackToCplxExtend, 10-36  
PadCurrent\_16x16\_MPEG4, 16-207  
PadCurrent\_8x8\_MPEG4, 16-207  
padding, of VOPs, 16-175  
PadMBGray\_MPEG4, 16-206  
PadMBHorizontal\_MPEG4, 16-204  
PadMBOpaque\_MPEG4, 16-237  
PadMBPartial\_MPEG4, 16-234  
PadMBTransparent\_MPEG4, 16-235  
PadMBVertical\_MPEG4, 16-205  
PadMV\_MPEG4, 16-209  
pattern recognition, 14-60  
Phase, 10-32  
PhasePack, 10-34  
Photo YCC color model, 6-14  
pixel order images, 2-18  
pixel planes, in MPEG-4 decoder output, 16-175  
planar order images, 2-18  
platforms supported, 1-2  
pOffset, 16-17  
point operations, 2-20  
ppBitStream, 16-17  
PredictIntra\_16x16\_H264, 16-350  
PredictIntra\_4x4\_H264, 16-348  
PredictIntraChroma8x8\_H264, 16-351  
PredictReconCoefIntra\_MPEG4, 16-202  
prefix  
    in data types, 2-5  
    in function names, 1-8  
PutIntraBlock, 16-141  
PutNonIntraBlock, 16-142  
PyramidFree, 14-46  
PyramidInitAlloc, 14-45  
PyramidLayerDown, 14-54  
PyramidLayerDownFree, 14-49  
PyramidLayerUp, 14-56

PyramidLayerUpFree, 14-51  
PyramidLayerUpInitAlloc, 14-49  
pyramids function  
    PyramidFree, 14-46  
    PyramidLayerDownFree, 14-49  
    PyramidLayerUp, 14-56  
    PyramidLayerUpFree, 14-51  
    PyramidLayerUpInitAlloc, 14-49  
PyrDown, 14-42  
PyrDownGetBufSize, 14-40  
PyrUp, 14-43  
PyrUpGetBufSize, 14-41

## Q

quality factor, in JPEG compression, 15-43  
QualityIndex, 11-80  
Quant\_MPEG2, 16-139  
QuantFwd8x8\_JPEG, 15-45  
QuantFwdRawTableInit\_JPEG, 15-43  
QuantFwdTableInit\_JPEG, 15-44  
QuantInter\_H263, 16-319  
QuantInter\_MPEG4, 16-253  
QuantInterGetSize\_MPEG4, 16-252  
QuantInterInit\_MPEG4, 16-251  
QuantIntra\_H263, 16-318  
QuantIntra\_MPEG2, 16-138  
QuantIntra\_MPEG4, 16-253  
QuantIntraGetSize\_MPEG4, 16-252  
QuantIntraInit\_MPEG4, 16-251  
QuantInv\_DV, 16-156  
QuantInv\_MPEG2, 16-128  
QuantInv8x8\_JPEG, 15-46  
QuantInvInter\_H263, 16-294  
QuantInvInter\_MPEG4, 16-212  
QuantInvInterGetSize\_MPEG4, 16-211  
QuantInvInterInit\_MPEG4, 16-210  
QuantInvIntra\_H263, 16-292  
QuantInvIntra\_MPEG2, 16-127  
QuantInvIntra\_MPEG4, 16-212

QuantInvIntraGetSize\_MPEG4, 16-211  
QuantInvIntraInit\_MPEG4, 16-210  
QuantInvTableInit\_JPEG, 15-46  
quantization functions, 15-43  
quantization parameters buffer, 16-182  
quantization parameters, of macroblocks, 16-178  
quantization table, 15-44  
QuantLuma8x8\_H264, 16-423  
QuantLuma8x8Inv\_H264, 16-430

## R

RCTFwd\_JPEG2K, 15-164  
RCTInv\_JPEG2K, 15-166  
ReconstructChromaInter4x4MB\_H264, 16-380  
ReconstructChromaInterMB\_H264, 16-374  
ReconstructChromaIntra4x4MB\_H264, 16-384  
ReconstructChromaIntraHalves4x4MB\_H264, 16-382  
ReconstructChromaIntraHalvesMB\_H264, 16-376  
ReconstructChromaIntraMB\_H264, 16-378  
ReconstructCoeffsInter\_H261, 16-274  
ReconstructCoeffsInter\_H263, 16-311  
ReconstructCoeffsInter\_MPEG4, 16-220  
ReconstructCoeffsIntra\_H261, 16-272  
ReconstructCoeffsIntra\_H263, 16-309  
ReconstructDCTBlock\_MPEG1, 16-119  
ReconstructDCTBlock\_MPEG2, 16-122  
ReconstructDCTBlockIntra\_MPEG1, 16-120  
ReconstructDCTBlockIntra\_MPEG2, 16-125  
ReconstructLumaInter4x4MB\_H264, 16-391  
ReconstructLumaInter8x8MB\_H264, 16-395  
ReconstructLumaInterMB\_H264, 16-386  
ReconstructLumaIntra16x16MB\_H264, 16-400, 16-402  
ReconstructLumaIntra4x4MB\_H264, 16-394  
ReconstructLumaIntra8x8MB\_H264, 16-398  
ReconstructLumaIntraHalf4x4MB\_H264, 16-392  
ReconstructLumaIntraHalf8x8MB\_H264, 16-397  
ReconstructLumaIntraHalfMB\_H264, 16-387  
ReconstructLumaIntraMB\_H264, 16-389

---

ReconstructPredFirstRow\_JPEG, 15-109  
ReconstructPredRow, 15-110  
RectStdDev, 11-20  
ReduceBits, 6-138  
region of interest, 2-20  
    in source and destination image, 2-21  
    offset, 2-22  
    size, 2-21  
related publications, 1-7  
Remap, 12-23  
Resample\_H263, 16-298  
Resize, 12-5  
ResizeCCRotate, 16-106  
ResizeCenter, 12-8  
ResizeShift, 12-17  
ResizeSqrPixel, 12-12  
ResizeSqrPixelGetBufSize, 12-15  
ResizeYUV422, 12-20  
RGB color model, 6-9  
RGB555ToYCbCr\_JPEG, 15-8  
RGB565ToYCbCr\_JPEG, 15-8  
RGBToCbYCr422, 6-65  
RGBToCbYCr422Gamma, 6-65  
RGBToGray, 6-123  
RGBToHLS, 6-105  
RGBToHSV, 6-111  
RGBToLUV, 6-95  
RGBToXYZ, 6-93  
RGBToY\_JPEG, 15-4  
RGBToYCbCr, 6-48  
RGBToYCbCr\_JPEG, 15-6  
RGBToYCbCr411LS\_MCU, 15-21  
RGBToYCbCr420, 6-75  
RGBToYCbCr422, 6-58  
RGBToYCbCr422LS\_MCU, 15-20  
RGBToYCbCr444LS\_MCU, 15-19  
RGBToYCC, 6-102  
RGBToYUV, 6-31  
RGBToYUV420, 6-38

RGBToYUV422, 6-34  
ROI processing, in geometric transforms, 12-4  
ROI. See region of interest  
Rotate, 12-26  
RotateCenter, 12-34  
row buffer, 16-182  
RShiftC, 5-72

## S

SAD16x16, 16-88  
SAD16x16Blocks4x4, 16-93  
SAD16x16Blocks8x8, 16-92  
SAD16x8, 16-89  
SAD4x4, 16-91  
SAD8x8, 16-90  
SampleDown411LS\_MCU, 15-66  
SampleDown422LS\_MCU, 15-65  
SampleDown444LS\_MCU, 15-64  
SampleDownH2V1\_JPEG, 15-55  
SampleDownH2V2\_JPEG, 15-56  
SampleDownRowH2V1\_Box\_JPEG, 15-57  
SampleDownRowH2V2\_Box\_JPEG, 15-58  
SampleLine, 4-39  
SampleUp411LS\_MCU, 15-69  
SampleUp444LS\_MCU, 15-67  
SampleUpH2V1\_JPEG, 15-59  
SampleUpH2V2\_JPEG, 15-61  
SampleUpRowH2V1\_Triangle\_JPEG, 15-62  
SampleUpRowH2V2\_Triangle\_JPEG, 15-63  
sampling functions, 15-54  
SBGRToYCoCg, 6-115  
SBGRToYCoCg\_Rev, 6-119  
Scale, 4-6  
scaling, of output results, 2-8  
ScanFwd, 16-101  
ScanInv, 16-99  
scanline padding, 12-4  
scanning  
    ScanFwd, 16-101

- ScanInv, 16-99
- segment, 16-147
- Set, 4-9
- Shear, 12-36
- SIMD instructions, 1-1
- SpatialInterpolation\_H263, 16-303
- Split422LS\_MCU, 15-70
- Sqr, 5-44
- SqrDiff16x16, 16-76
- SqrDiff16x16B, 16-77
- SqrDistanceFull\_Norm, 11-84
- SqrDistanceSame\_Norm, 11-86
- SqrDistanceValid\_Norm, 11-88
- SqrIntegral, 11-9
- Sqrt, 5-46
- SSD4x4, 16-79
- SSD8x8, 16-78
- statistics functions
  - computing image moments, 11-45
  - context initializing, 11-42
  - deallocation, 11-43
  - HistogramEven, 11-27
  - HistogramRange, 11-24
  - image norms
    - infinity norm, 11-54
    - infinity norm of differences, 11-62
    - L1- norm, 11-56
    - L1- norm of differences, 11-65
    - L2- norm, 11-60
    - L2- norm of differences, 11-68
    - relative error, 11-71
  - integral of pixel squares, 11-9
  - integral representation, 11-7
  - maximum, 11-34
  - mean and stdev value, 11-18
  - mean value, 11-15
  - minimum, 11-31
  - minimum and maximum, 11-37
  - pixels count, 11-30
  - retrieving central moments, 11-50
  - retrieving Hu moments, 11-52
  - retrieving spatial moments, 11-47
  - standard deviation of integral images, 11-20

- standard deviation of tilted integral images, 11-22
- sum, 11-5
- tilted integral of pixel squares, 11-13
- tilted integral representation, 11-11
- status codes, 2-9
- status information function
  - ippGetStatusString, 3-4
- step
  - through the buffer, 2-21
- Streaming SIMD Extensions, 1-1
- structures in IPP, 2-13
- Sub, 5-27
- Sub128\_JPEG, 15-52
- Sub16x16, 16-73, 16-74
- Sub8x8, 16-73
- SubC, 5-30
- SubSAD8x8, 16-74
- Successive Elimination Algorithm, 16-243
- Sum, 11-5
- SumNorm\_VOP\_MPEG4, 16-242
- SumsDiff16x16Blocks4x4, 16-96
- SumsDiff8x8Blocks4x4, 16-98
- SumWindowColumn, 9-11
- SumWindowRow, 9-10
- superblock, 16-146
- support functions, 3-1
- SwapChannels, 4-30

## T

- Threshold, 7-2
- threshold functions, 7-2
- Threshold\_GT, 7-5
- Threshold\_GTVal, 7-11
- Threshold\_LT, 7-7
- Threshold\_LTVal, 7-13
- Threshold\_LTValGTVal, 7-16
- Threshold\_Val, 7-9
- tiled image processing, 2-24
- Tilted RectStdDev, 11-22

---

TiltedHaarClassifierInitAlloc, 14-63  
TiltedIntegral, 11-11  
TiltedSqrIntegral, 11-13  
TransformDequantChromaDC\_H264, 16-340  
TransformDequantChromaDC\_SISP\_H264, 16-347  
TransformDequantLumaDC\_H264, 16-339  
TransformLuma8x8Fwd\_H264, 16-422  
TransformLuma8x8InvAddPred\_H264, 16-431  
TransformPrediction\_H264, 16-345  
TransformQuantChromaDC\_H264, 16-417  
TransformQuantLumaDC\_H264, 16-419  
TransformQuantResidual\_H264\_16s\_C1I, 16-421  
transparent status buffer, 16-182  
transparent status, of a macroblock, 16-178  
Transpose, 4-28  
TransRecBlockCoef\_inter\_MPEG4, 16-260  
TransRecBlockCoef\_intra\_MPEG4, 16-261

## U

UndistortGetSize, 14-72  
UndistortRadial, 14-73  
UniDirWeightBlock\_H264, 16-369  
uniform chromaticity scale diagram, 6-22  
universal pyramids, 14-45  
UpdateMotionHistory, 14-33  
UpsampleFour\_H263, 16-300  
UpsampleFour8x8\_H263, 16-302

## V

Variance16x16, 16-85, 16-87  
VarMean8x8, 16-80  
VarMeanDiff16x16, 16-82  
VarMeanDiff16x8, 16-83  
version information function  
    GetLibVersion, 3-2  
version information, for JPEG library, 15-2  
vertical sampling, in a JPEG codec, 15-54  
video coding color conversion, 16-104

video encoder functions, Block Encoding  
    TransRecBlockCoef\_inter\_MPEG4, 16-260  
    TransRecBlockCoef\_intra\_MPEG4, 16-261  
video encoder functions, motion estimation  
    BlockMatch\_Integer\_16x16\_MVFAST, 16-246  
    BlockMatch\_Integer\_16x16\_SEA, 16-243  
    ComputeTextureErrorBlock, 16-250  
    ComputeTextureErrorBlock\_SAD, 16-249  
    MeanAbsDev16x16, 16-86  
    MotionEstimation\_16x16\_MVFAST, 16-248  
    MotionEstimation\_16x16\_SEA, 16-245  
    SumNorm\_VOP\_MPEG4, 16-242  
video encoder functions, MV Encoding  
    EncodeMV\_MPEG4, 16-264  
    FindMVPred\_MPEG4, 16-263  
video encoder functions, VLC Encoding  
    EncodeVLCZigzag\_Inter\_MPEG4, 16-259  
    EncodeVLCZigzag\_IntraACVLC\_MPEG4, 16-258  
    EncodeVLCZigzag\_IntraDCVLC\_MPEG4, 16-258  
video plane buffers, 16-181  
video processing functions  
    general color conversion  
        CbYCr422ToYCbCr420\_Rotate, 16-104  
        ResizeCCRotate, 16-106  
video processing functions, general  
    DeinterlaceFilterTriangle, 16-109  
    ZigzagInvClassical\_Compact, 16-102  
    ZigzagInvHorizontal\_Compact, 16-102  
    ZigzagInvVertical\_Compact, 16-102  
VOL, video object layer, 16-175  
VOP, video object plane, 16-173

## W

WarpAffine, 12-41  
WarpAffineBack, 12-45  
WarpAffineQuad, 12-48  
WarpBilinear, 12-66  
WarpBilinearBack, 12-69  
WarpBilinearQuad, 12-72  
WarpChroma\_MPEG4, 16-190  
WarpGetSize\_MPEG4, 16-188  
WarpInit\_MPEG4, 16-187



WarpLuma\_MPEG4, 16-189  
WarpPerspective, 12-54  
WarpPerspectiveBack, 12-57  
WarpPerspectiveQuad, 12-60  
wavelet transform functions  
    buffer size calculation, 13-7, 13-16  
    deallocation, 13-7, 13-15  
    forward transform, 13-8  
    initialization, 13-5, 13-13  
    inverse transform, 13-17  
wavelet transforms, for lossless compression, 15-114  
WeightedAverage\_H264, 16-368  
WTFwd, 13-8  
WTFwd\_B53\_JPEG2K, 15-136  
WTFwdCol\_B53\_JPEG2K, 15-120  
WTFwdCol\_D97\_JPEG2K, 15-129  
WTFwdColLift\_B53\_JPEG2K, 15-121  
WTFwdColLift\_D97\_JPEG2K, 15-130  
WTFwdFree, 13-7  
WTFwdGetBufSize, 13-7  
WTFwdInitAlloc, 13-5  
WTFwdRow\_B53\_JPEG2K, 15-117  
WTFwdRow\_D97\_JPEG2K, 15-126  
WTInv, 13-17  
WTInv\_B53\_JPEG2K, 15-138  
WTInv\_D97\_JPEG2K, 15-142  
WTInvCol\_B53\_JPEG2K, 15-123  
WTInvCol\_D97\_JPEG2K, 15-132  
WTInvColLift\_B53\_JPEG2K, 15-124  
WTInvColLift\_D97\_JPEG2K, 15-133  
WTInvFree, 13-15  
WTInvGetBufSize, 13-16  
WTInvInitAlloc, 13-13  
WTInvRow\_B53\_JPEG2K, 15-118  
WTInvRow\_D97\_JPEG2K, 15-127

## X

Xor, 5-65  
XorC, 5-67

XYZToRGB, 6-94

## Y

YCbC422ToBGR555Dither, 6-74  
YCbC422ToBGR565Dither, 6-74  
YCbCr411, 6-171  
YCbCr411ToBGR, 6-89  
YCbCr411ToBGR555, 6-92  
YCbCr411ToBGR555LS\_MCU, 15-39  
YCbCr411ToBGR565, 6-92  
YCbCr411ToBGR565LS\_MCU, 15-39  
YCbCr411ToBGR565LS\_MCU, 15-38  
YCbCr411ToRGBLS\_MCU, 15-33  
YCbCr411ToYCbCr420, 6-175  
YCbCr411ToYCbCr422, 6-173  
YCbCr411ToYCrCb420, 6-177  
YCbCr411ToYCrCb422, 6-174  
YCbCr420, 6-156  
YCbCr420ToBGR, 6-81  
YCbCr420ToBGR444Dither, 6-85  
YCbCr420ToBGR555Dither, 6-85  
YCbCr420ToBGR565Dither, 6-85  
YCbCr420ToCbYCr422, 6-161  
YCbCr420ToRGB, 6-76  
YCbCr420ToRGB444Dither, 6-79  
YCbCr420ToRGB555Dither, 6-79  
YCbCr420ToRGB565Dither, 6-79  
YCbCr420ToYCbCr411, 6-165  
YCbCr420ToYCbCr422, 6-157  
YCbCr420ToYCbCr422\_Filter, 6-159  
YCbCr420ToYCrCb420, 6-162  
YCbCr420ToYCrCb420\_Filter, 6-163  
YCbCr422, 6-141  
YCbCr422ToBGR, 6-62  
YCbCr422ToBGR444, 6-72  
YCbCr422ToBGR444Dither, 6-74  
YCbCr422ToBGR555, 6-72  
YCbCr422ToBGR555LS\_MCU, 15-37

---

YCbCr422ToBGR565, 6-72	YCbCrToRGB565Dither, 6-54
YCbCr422ToBGR565LS_MCU, 15-37	YCCK color model, 6-13
YCbCr422ToBGR565LS_MCU, 15-36	YCCK444ToCMYKLS_MCU, 15-40
YCbCr422ToCbYCr422, 6-143	YCCKToCMYK_JPEG, 15-16
YCbCr422ToRGB, 6-59	YCCKToCMYK411LS_MCU, 15-42
YCbCr422ToRGB444, 6-69	YCCKToCMYK422LS_MCU, 15-41
YCbCr422ToRGB444Dither, 6-71	YCCToRGB, 6-104
YCbCr422ToRGB555, 6-69	YCrCr color model, 6-13
YCbCr422ToRGB555Dither, 6-71	YCoCg color model, 6-16
YCbCr422ToRGB565, 6-69	YCoCgToBGR, 6-116
YCbCr422ToRGB565Dither, 6-71	YCoCgToBGR_Rev, 6-120
YCbCr422ToRGBLS_MCU, 15-32	YCoCgToSBGR, 6-117
YCbCr422ToYCbCr411, 6-147	YCoCgToSBGR_Rev, 6-121
YCbCr422ToYCbCr420, 6-144	YCrCb411ToYCbCr422_5MBDV, 16-162
YCbCr422ToYCrCb420, 6-146	YCrCb411ToYCbCr422_EdgeDV, 16-163
YCbCr422ToYCrCb422, 6-142	YCrCb411ToYCbCr422_ZoomOut2_5MBDV, 16-162
YCbCr444ToBGR555LS_MCU, 15-35	YCrCb411ToYCbCr422_ZoomOut2_EdgeDV, 16-163
YCbCr444ToBGR565LS_MCU, 15-35	YCrCb411ToYCbCr422_ZoomOut4_5MBDV, 16-162
YCbCr444ToBGR565LS_MCU, 15-34	YCrCb411ToYCbCr422_ZoomOut4_EdgeDV, 16-163
YCbCr444ToRGBLS_MCU, 15-31	YCrCb411ToYCbCr422_ZoomOut8_5MBDV, 16-162
YCbCrToBGR, 6-51	YCrCb411ToYCbCr422_ZoomOut8_EdgeDV, 16-163
YCbCrToBGR_JPEG, 15-11	YCrCb420ToCbYCr422, 6-168
YCbCrToBGR444, 6-55	YCrCb420ToYCbCr411, 6-170
YCbCrToBGR444Dither, 6-57	YCrCb420ToYCbCr420, 6-169
YCbCrToBGR555, 6-55	YCrCb420ToYCbCr422, 6-166
YCbCrToBGR555_JPEG, 15-13	YCrCb420ToYCbCr422_5MBDV, 16-164
YCbCrToBGR555Dither, 6-57	YCrCb420ToYCbCr422_Filter, 6-167
YCbCrToBGR565, 6-55	YCrCb420ToYCbCr422_ZoomOut2_5MBDV, 16-164
YCbCrToBGR565_JPEG, 15-13	YCrCb420ToYCbCr422_ZoomOut4_5MBDV, 16-164
YCbCrToBGR565Dither, 6-57	YCrCb420ToYCbCr422_ZoomOut8_5MBDV, 16-164
YCbCrToRGB, 6-50	YCrCb422ToYCbCr411, 6-151
YCbCrToRGB_JPEG, 15-7	YCrCb422ToYCbCr420, 6-150
YCbCrToRGB444, 6-52	YCrCb422ToYCbCr422, 6-149
YCbCrToRGB444Dither, 6-54	YCrCb422ToYCbCr422_5MBDV, 16-166
YCbCrToRGB555, 6-52	YCrCb422ToYCbCr422_ZoomOut2_5MBDV, 16-166
YCbCrToRGB555_JPEG, 15-9	YCrCb422ToYCbCr422_ZoomOut4_5MBDV, 16-166
YCbCrToRGB555Dither, 6-54	YCrCb422ToYCbCr422_ZoomOut8_5MBDV, 16-166
YCbCrToRGB565, 6-52	YUV color model, 6-11
YCbCrToRGB565_JPEG, 15-9	YUV420ToBGR, 6-41

YUV420ToBGR444, 6-46  
YUV420ToBGR444Dither, 6-47  
YUV420ToBGR555, 6-46  
YUV420ToBGR555Dither, 6-47  
YUV420ToBGR565, 6-46  
YUV420ToBGR565Dither, 6-47  
YUV420ToRGB, 6-39  
YUV420ToRGB444, 6-42  
YUV420ToRGB444Dither, 6-43  
YUV420ToRGB555, 6-42  
YUV420ToRGB555Dither, 6-43  
YUV420ToRGB565, 6-42  
YUV420ToRGB565Dither, 6-43  
YUV422ToRGB, 6-36  
YUVToRGB, 6-33

## **Z**

zigzag order of elements, 15-44  
ZigzagFwd8x8, 4-40  
ZigzagInv8x8, 4-41  
ZigzagInvClassical\_Compact, 16-102  
ZigzagInvHorizontal\_Compact, 16-102  
ZigzagInvVertical\_Compact, 16-102